pyiron Documentation

Release 0.2.2

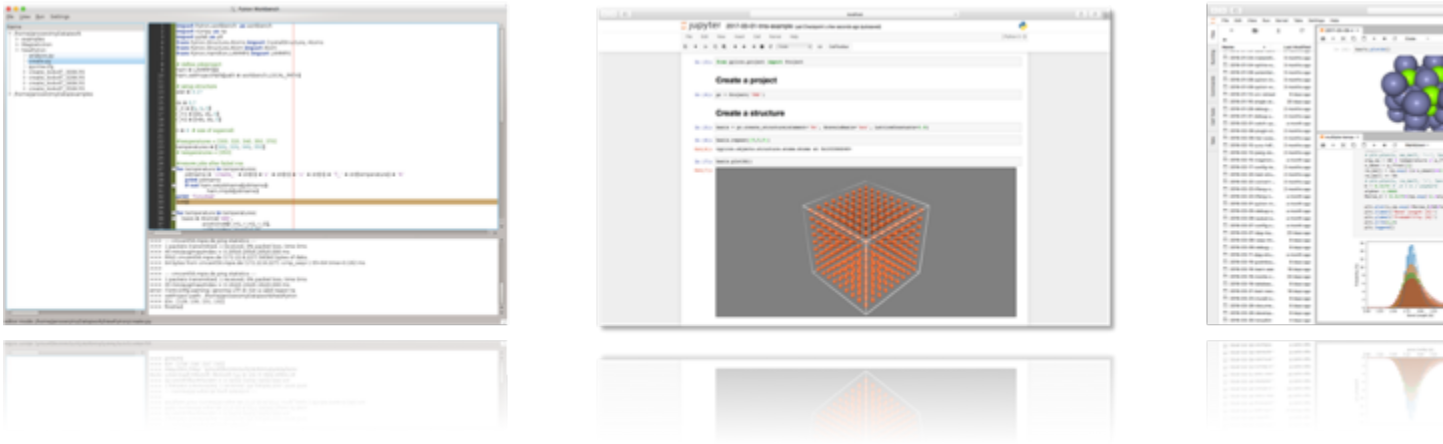
Max-Planck-Institut für Eisenforschung GmbH - Computational Ma

Feb 23, 2020

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 Insitut für Eisenforschung (Max Planck Insitute 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 1

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 2

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.

CHAPTER 3

News

- **20th December 2019:** Announcement of the 1st pyiron workshop in Bochum (Germany) from the 31st of March to the 2nd of April 2020.
- **09th November 2019:** pyiron was downloaded over 10000 times on conda-forge and gained attention with over 50 stars on github.
- **10th October 2019:** pyiron 0.2.9 released.
- **20th June 2019:** pyiron was downloaded over 5000 times on conda-forge and 70% of our code are covered with unit tests.
- **10th May 2019:** pyiron documentation hosted on readthedocs.org.
- **24th March 2019:** pyiron 0.2.2 released.
- **15th March 2019:** pyiron paper available as open access .
- **20th January 2019:** pyiron 0.2.1 released.
- **15th December 2019:** pyiron was downloaded over 2000 times on conda-forge .
- **21st November 2018:** pyiron 0.2.0 released.
- **2nd August 2018:** pyiron 0.1.8 released.
- **21st July 2018:** pyiron paper accepted.
- **20th July 2018:** pyiron 0.1.7 released.
- **25th May 2018:** pyiron 0.1.5 released.
- **11th May 2018:** pyiron 0.1.3 published on conda-forge install pyiron using: `conda install -c conda-forge pyiron`
- **07th May 2018:** pyiron paper submitted
- **05th April 2018:** test pyiron on mybinder.org (beta)
- **27th March 2018:** pyiron is available on anaconda.org install pyiron using: `conda install -c pyiron -c conda-forge pyiron`

- **27th February 2018:** pyiron is available on pypi.org install pyiron using: `pip install pyiron`
- **05th December 2017:** The pyiron website goes online.

CHAPTER 4

Citing

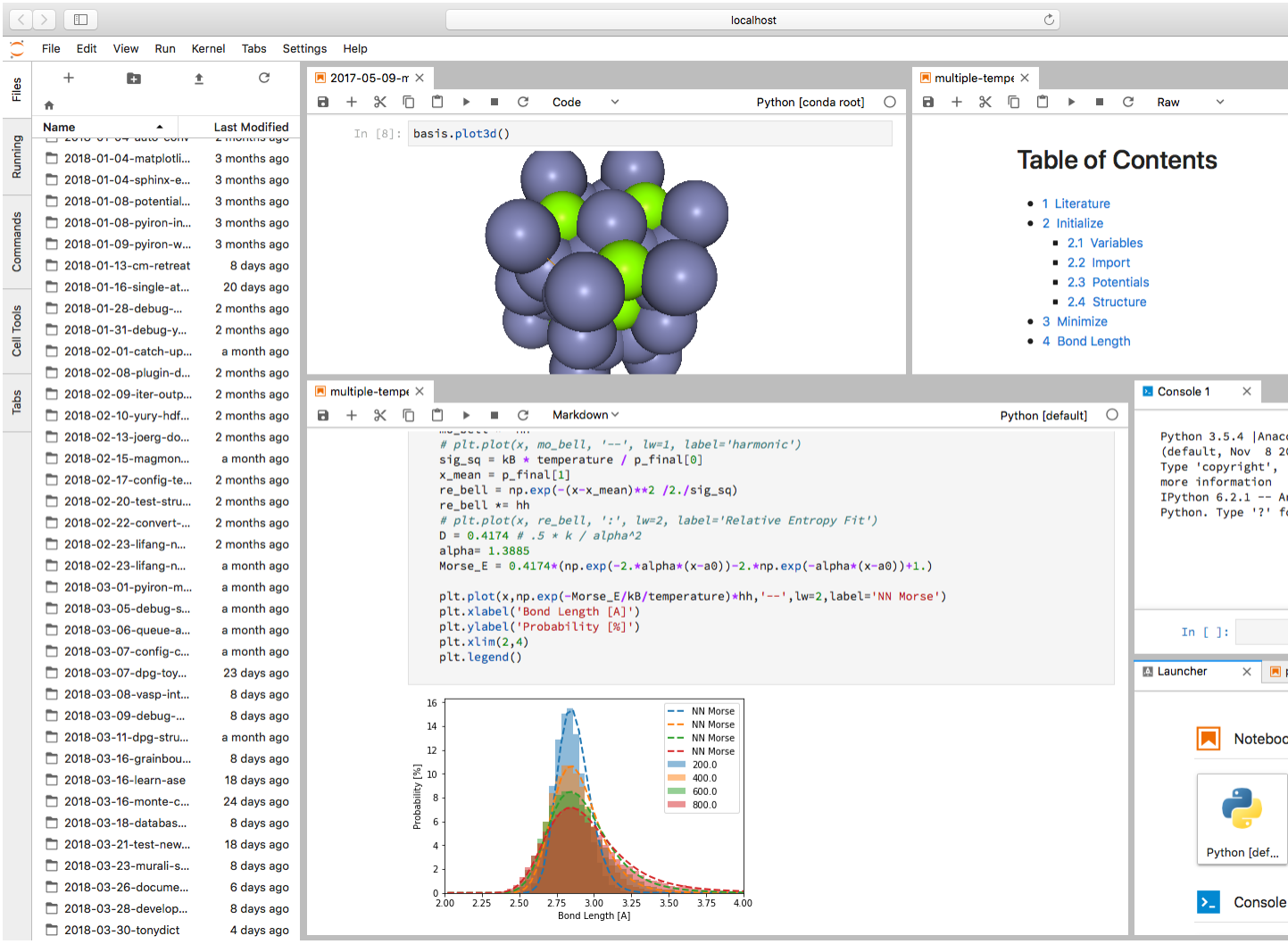
If you use pyiron in your research, please consider citing the following work:

```
@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.

4.1 About

4.1.1 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 Insitut für Eisenforschung (Max Planck Insitute 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.

4.1.2 Getting Help

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

4.1.3 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

4.2 Installation

Note: **Before you install:** We provide various levels of environments to test pyiron:

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

4.2.1 Workstation Installation (recommeded)

Requirements

When you start to develop your own simulation protocols we recommend a local installation. Inside the pyiron anaconda repository we provide precompiled executables for Linux, Mac OS X and Windows with Python 2.7, 3.5, 3.6 and 3.7 and the other packages are available inside the conda-forge channel.

Install pyiron package

As pyiron is written in Python you can install pyiron either via anaconda (recommended) or via pip.

Install via anaconda (recommended):

To install anaconda you can download the anaconda distribution. Following the installation update to the latest version of conda from conda-forge.

```
conda update -c conda-forge conda
```

After the update of the anaconda environment you can install pyiron using:

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

Install via pip:

pip is installed on Linux and Mac Os X by default and is included in most Python distributions. To install pyiron via pip type:

```
pip install pyiron
```

While the anaconda installation already includes the lammps executable, the pip installation requires the user to include a lammps executable named `lmp_serial` for Linux and Mac Os X or `lmp_serial.exe` for windows in their `PATH`.

Visualization

In addition to the pyiron package we recommend installing the NGLview visualization framework.

Stable version – for jupyter notebooks (recommended):

```
conda install -c conda-forge nglview
jupyter nbextension install nglview --py --sys-prefix
jupyter nbextension enable nglview --py --sys-prefix
```

Stable version – for jupyter lab

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

Simulation code: Lammps

pyiron supports the simulation codes VASP for DFT calculation and Lammps for molecular dynamics calculation. While VASP requires a separate license and therefore has to be configured by the user, Lammps is available as open-source code and can be installed from anaconda.

For Linux and Mac Os X (for Python 2.7, 3.5, 3.6 and 3.7):

```
conda install -c conda-forge lammps
```

For windows:

```
conda install -c pyiron lammps
```

Configuration

After the installation of pyiron we need to configure pyiron. The default configuration can be generated automatically. In the terminal, start a new Python session 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()
```

The configuration does the following steps in the background:

- Create an `~/.pyiron` config file – with the default settings (for simple installations)
- Create an `~/pyiron/projects` directory – pyiron can only execute calculation within this project directory to prevent any interference, with other tools or simulation management solutions.
- Create an `~/pyiron/resources` directory – this directory includes the link to the executables and potentials, sorted by code. The potentials for lammps are inside `pyiron_lammps` and those for vasp can be placed in `pyiron_vasp`.

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 visualize 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 step

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

4.2.2 Computer Cluster (HPC)

While the local Installation is designed to scale beyond a single workstation, further multi user extensions are required like:

- Jupyterhub for managing multiple Jupyter Sessions.
- PostgreSQL database for scalability.
- Queuing system for job management.
- Access Control lists for sharing files between users.

For further details please open a support request.

4.2.3 Mybinder.org (beta)

Warning: Mybinder.org is currently in beta stage, it should not take longer than a minute to load. We are sorry for the inconvenience.

You can test pyiron on Mybinder.org (beta), without the need of a local installation. This installation comes with the following limitations:

- No VASP license, DFT calculation can be imported and loaded but the execution is disabled.
- No visualization of atomistic structures using NGLview.
- Only temporary data storage, when you leave your session on Mybinder.org (beta) the environment is reset.

The Mybinder service is the most flexible way to test pyiron and get a first impression. Start pyiron on MyBinder.org to test your first pyiron examples.

4.2.4 Docker (for demonstration)

Commonly it is easier to install pyiron directly using anaconda following the Local Installation (Workstation) instead of installing Docker. If you already setup Docker on your system, you might still be interested in downloading the pyiron container. While Mybinder.org (beta) is based on a similar Docker image, running the Docker image locally enables more flexibility. In particular the graphical user interface is fully supported in this version. Still the following limitations remain:

- No VASP license, DFT calculation can be imported and loaded but the execution is disabled.
- Only temporary data storage, when you shutdown your Docker instance the environment is reset.

This installation of pyiron is most suitable for presentations. After the local installation of Docker there are two versions to choose from stable version based on jupyter notebooks and the latest beta version based on jupyter lab. For both versions the first command downloads the image from Dockerhub and the second command executes it locally.

Docker image with jupyter notebook (stable)

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

```
docker run -i -t -p 8888:8888 pyiron/pyiron /bin/bash -c "/opt/conda/bin/jupyter␣
↪notebook --notebook-dir=/home/pyiron/ --ip='*' --port=8888"
```

Docker image with jupyter lab (beta)

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

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

Connect

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 also the Docker image comes with the examples preinstalled.

4.3 Tutorials

4.3.1 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.project 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']}
```

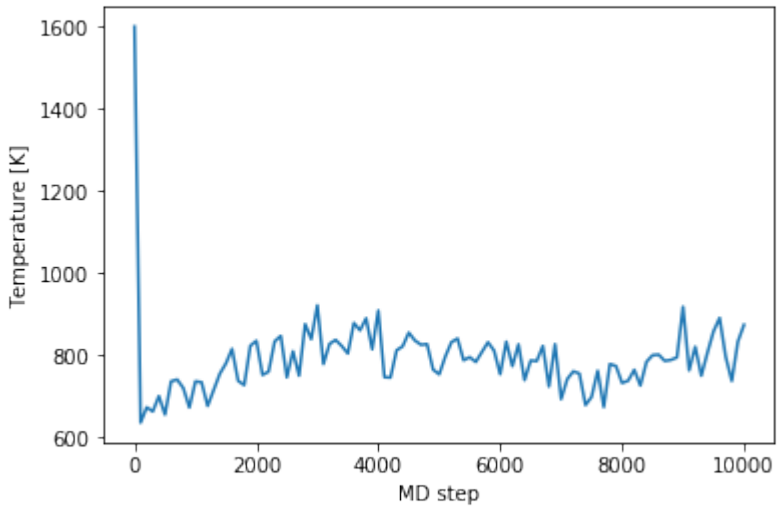
An animated 3d plot of the MD trajectories is created by:

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

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

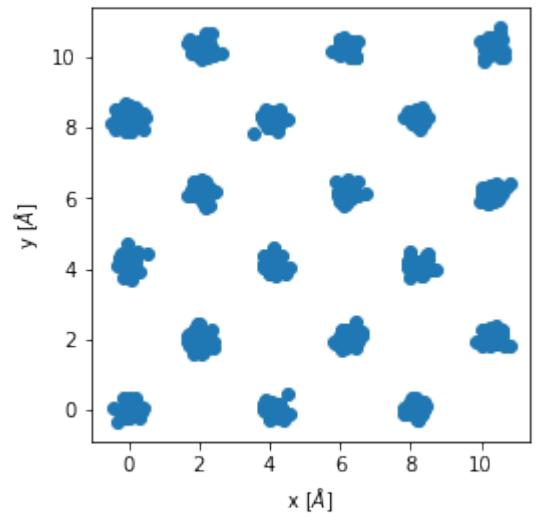
To analyze the temperature evolution we plot it as function of the MD step.

[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 [$\AA$]')
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
```

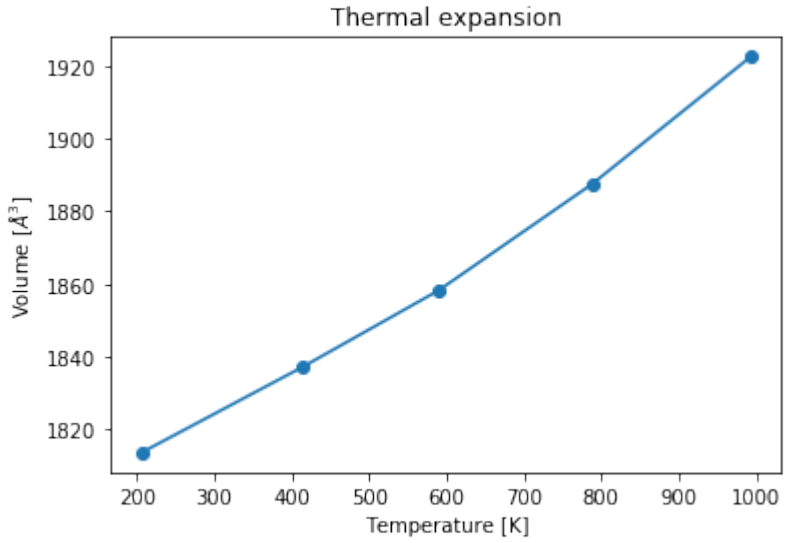
[16]:
```
['Al_T600K', 'Al_T800K', 'Al_T1000K', 'Al_T200K', 'Al_T400K']
```

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.

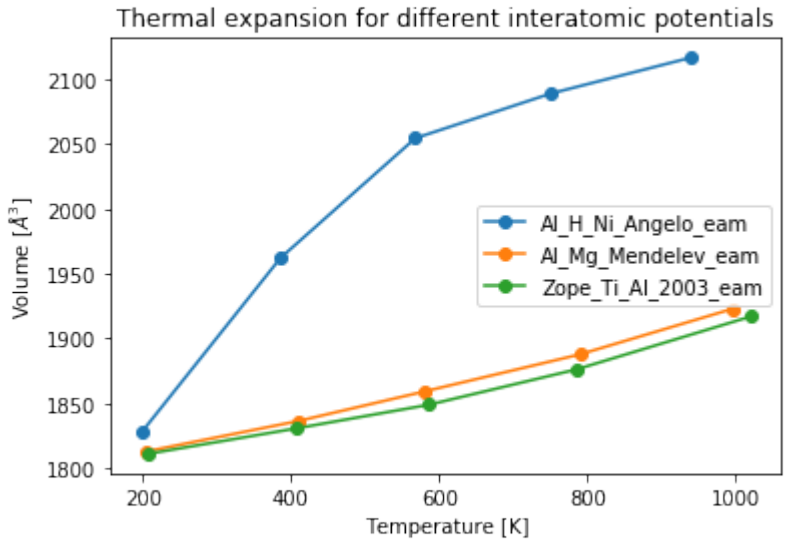
[21]:
```
for pot in pot_lst:
    print ('Interatomic potential used: ',pot)
    pr_pot = pr.create_group(pot)
    for temperature in np.arange(200, 1200, 200):
        job = pr_pot.create_job(pr.job_type.Lammps,
                                'Al_T{}K'.format(int(temperature)))
        job.structure = 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 [$\AA^3$]');
```



[ ]:

4.3.2 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. In addition we remove all jobs which might exist in the project before to have a clean project for our example.

[2]:
```
pr = Project(path='thermo')
# pr.remove_jobs(recursive=True)
```

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()
```

```
_ColormakerRegistry()
```

```
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(0.97, 1.03, 7):
    strain_str = str(strain).replace('.', '_')
    job_vasp_strain = pr.create_job(job_type=pr.job_type.GpawJob, 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()
```

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	subjob	projectpath	\
6	3601535	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	
38	3601804	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	
45	3602090	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	
8	3602359	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	

(continues on next page)

(continued from previous page)

```
39  3602595  finished            None  gpaw_1_01  /gpaw_1_01    /cmmc/u/
0   3602869  finished            None  gpaw_1_02  /gpaw_1_02    /cmmc/u/
4   3603146  finished            None  gpaw_1_03  /gpaw_1_03    /cmmc/u/
37  3603487  finished            None  gpaw_0_97  /gpaw_0_97    /cmmc/u/
5   3603765  finished            None  gpaw_0_98  /gpaw_0_98    /cmmc/u/
44  3604021  finished            None  gpaw_0_99  /gpaw_0_99    /cmmc/u/
2   3604219  finished            None   gpaw_1_0   /gpaw_1_0    /cmmc/u/
7   3604443  finished            None  gpaw_1_01  /gpaw_1_01    /cmmc/u/
3   3604655  finished            None  gpaw_1_02  /gpaw_1_02    /cmmc/u/
1   3604768  finished            None  gpaw_1_03  /gpaw_1_03    /cmmc/u/
40  3604824  finished            None  gpaw_0_97  /gpaw_0_97    /cmmc/u/
42  3604871  finished            None  gpaw_0_98  /gpaw_0_98    /cmmc/u/
43  3604875  finished            None  gpaw_0_99  /gpaw_0_99    /cmmc/u/
48  3604880  finished            None   gpaw_1_0   /gpaw_1_0    /cmmc/u/
9   3604884  finished            None  gpaw_1_01  /gpaw_1_01    /cmmc/u/
10  3604887  finished            None  gpaw_1_02  /gpaw_1_02    /cmmc/u/
19  3604890  finished            None  gpaw_1_03  /gpaw_1_03    /cmmc/u/
41  3604896  finished            None  gpaw_0_97  /gpaw_0_97    /cmmc/u/
46  3604899  finished            None  gpaw_0_98  /gpaw_0_98    /cmmc/u/
47  3604903  finished            None  gpaw_0_99  /gpaw_0_99    /cmmc/u/
11  3604907  finished            None   gpaw_1_0   /gpaw_1_0    /cmmc/u/
12  3604910  finished            None  gpaw_1_01  /gpaw_1_01    /cmmc/u/
13  3604911  finished            None  gpaw_1_02  /gpaw_1_02    /cmmc/u/
14  3604913  finished            None  gpaw_1_03  /gpaw_1_03    /cmmc/u/
15  3604914  finished            None  gpaw_0_97  /gpaw_0_97    /cmmc/u/
16  3604915  finished            None  gpaw_0_98  /gpaw_0_98    /cmmc/u/
17  3604916  finished            None  gpaw_0_99  /gpaw_0_99    /cmmc/u/
18  3604917  finished            None   gpaw_1_0   /gpaw_1_0    /cmmc/u/
20  3604918  finished            None  gpaw_1_01  /gpaw_1_01    /cmmc/u/
21  3604919  finished            None  gpaw_1_02  /gpaw_1_02    /cmmc/u/
22  3604920  finished            None  gpaw_1_03  /gpaw_1_03    /cmmc/u/
23  3604921  finished            None  gpaw_0_97  /gpaw_0_97    /cmmc/u/
24  3604922  finished            None  gpaw_0_98  /gpaw_0_98    /cmmc/u/
25  3604923  finished            None  gpaw_0_99  /gpaw_0_99    /cmmc/u/
26  3604924  finished            None   gpaw_1_0   /gpaw_1_0    /cmmc/u/
27  3604925  finished            None  gpaw_1_01  /gpaw_1_01    /cmmc/u/
28  3604926  finished            None  gpaw_1_02  /gpaw_1_02    /cmmc/u/
29  3604927  finished            None  gpaw_1_03  /gpaw_1_03    /cmmc/u/
30  3604928  finished            None  gpaw_0_97  /gpaw_0_97    /cmmc/u/
31  3604929  finished            None  gpaw_0_98  /gpaw_0_98    /cmmc/u/
32  3604930  finished            None  gpaw_0_99  /gpaw_0_99    /cmmc/u/
33  3604931  finished            None   gpaw_1_0   /gpaw_1_0    /cmmc/u/
34  3604933  finished            None  gpaw_1_01  /gpaw_1_01    /cmmc/u/
35  3604934  finished            None  gpaw_1_02  /gpaw_1_02    /cmmc/u/
36  3604935  finished            None  gpaw_1_03  /gpaw_1_03    /cmmc/u/

                                                                 project  \
6             janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
38            janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
45            janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
8             janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
39            janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
0             janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
4             janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
37  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
5   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
44  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
```

(continues on next page)

(continued from previous page)

```
2   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
7   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
3   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
1   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
40  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
42  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
43  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
48  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
9   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
10  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
19  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
41  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
46  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
47  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
11  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
12  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
13  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
14  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
15  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
16  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
17  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
18  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
20  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
21  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
22  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
23  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
24  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
25  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
26  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
27  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
28  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
29  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
30  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
31  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
32  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
33  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
34  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
35  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
36  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/

                    timestart timestop totalcputime        computer hamilton  \
6  2019-09-04 13:50:31.285688     None         None  janj@cmmc001#1  GpawJob
38 2019-09-04 13:50:42.970024     None         None  janj@cmmc001#1  GpawJob
45 2019-09-04 13:50:52.350144     None         None  janj@cmmc001#1  GpawJob
8  2019-09-04 13:51:01.413156     None         None  janj@cmmc001#1  GpawJob
39 2019-09-04 13:51:10.251511     None         None  janj@cmmc001#1  GpawJob
0  2019-09-04 13:51:20.041600     None         None  janj@cmmc001#1  GpawJob
4  2019-09-04 13:51:29.741836     None         None  janj@cmmc001#1  GpawJob
37 2019-09-04 13:51:42.796905     None         None  janj@cmmc001#1  GpawJob
5  2019-09-04 13:51:53.088836     None         None  janj@cmmc001#1  GpawJob
44 2019-09-04 13:52:02.979283     None         None  janj@cmmc001#1  GpawJob
2  2019-09-04 13:52:11.921097     None         None  janj@cmmc001#1  GpawJob
7  2019-09-04 13:52:21.335934     None         None  janj@cmmc001#1  GpawJob
3  2019-09-04 13:52:31.009130     None         None  janj@cmmc001#1  GpawJob
1  2019-09-04 13:52:40.320534     None         None  janj@cmmc001#1  GpawJob
40 2019-09-04 13:52:50.470563     None         None  janj@cmmc001#1  GpawJob
42 2019-09-04 13:52:59.954797     None         None  janj@cmmc001#1  GpawJob
```

(continues on next page)

(continued from previous page)

```
43 2019-09-04 13:53:09.963464     None         None  janj@cmmc001#1  GpawJob
48 2019-09-04 13:53:20.141701     None         None  janj@cmmc001#1  GpawJob
9  2019-09-04 13:53:29.324131     None         None  janj@cmmc001#1  GpawJob
10 2019-09-04 13:53:38.550641     None         None  janj@cmmc001#1  GpawJob
19 2019-09-04 13:53:48.099532     None         None  janj@cmmc001#1  GpawJob
41 2019-09-04 13:53:59.152323     None         None  janj@cmmc001#1  GpawJob
46 2019-09-04 13:54:08.057464     None         None  janj@cmmc001#1  GpawJob
47 2019-09-04 13:54:17.516512     None         None  janj@cmmc001#1  GpawJob
11 2019-09-04 13:54:26.874849     None         None  janj@cmmc001#1  GpawJob
12 2019-09-04 13:54:37.345194     None         None  janj@cmmc001#1  GpawJob
13 2019-09-04 13:54:48.785761     None         None  janj@cmmc001#1  GpawJob
14 2019-09-04 13:55:00.556380     None         None  janj@cmmc001#1  GpawJob
15 2019-09-04 13:55:10.828970     None         None  janj@cmmc001#1  GpawJob
16 2019-09-04 13:55:19.451476     None         None  janj@cmmc001#1  GpawJob
17 2019-09-04 13:55:28.235999     None         None  janj@cmmc001#1  GpawJob
18 2019-09-04 13:55:36.912405     None         None  janj@cmmc001#1  GpawJob
20 2019-09-04 13:55:46.777440     None         None  janj@cmmc001#1  GpawJob
21 2019-09-04 13:55:55.186420     None         None  janj@cmmc001#1  GpawJob
22 2019-09-04 13:56:05.185718     None         None  janj@cmmc001#1  GpawJob
23 2019-09-04 13:56:14.138209     None         None  janj@cmmc001#1  GpawJob
24 2019-09-04 13:56:22.072544     None         None  janj@cmmc001#1  GpawJob
25 2019-09-04 13:56:30.773140     None         None  janj@cmmc001#1  GpawJob
26 2019-09-04 13:56:38.738514     None         None  janj@cmmc001#1  GpawJob
27 2019-09-04 13:56:46.853680     None         None  janj@cmmc001#1  GpawJob
28 2019-09-04 13:56:55.404483     None         None  janj@cmmc001#1  GpawJob
29 2019-09-04 13:57:03.940368     None         None  janj@cmmc001#1  GpawJob
30 2019-09-04 13:57:13.347542     None         None  janj@cmmc001#1  GpawJob
31 2019-09-04 13:57:21.459612     None         None  janj@cmmc001#1  GpawJob
32 2019-09-04 13:57:29.623085     None         None  janj@cmmc001#1  GpawJob
33 2019-09-04 13:57:37.903535     None         None  janj@cmmc001#1  GpawJob
34 2019-09-04 13:57:46.032614     None         None  janj@cmmc001#1  GpawJob
35 2019-09-04 13:57:54.280901     None         None  janj@cmmc001#1  GpawJob
36 2019-09-04 13:58:02.411374     None         None  janj@cmmc001#1  GpawJob

   hamversion parentid masterid
6        None     None     None
38       None     None     None
45       None     None     None
8        None     None     None
39       None     None     None
0        None     None     None
4        None     None     None
37       None     None     None
5        None     None     None
44       None     None     None
2        None     None     None
7        None     None     None
3        None     None     None
1        None     None     None
40       None     None     None
42       None     None     None
43       None     None     None
48       None     None     None
9        None     None     None
10       None     None     None
19       None     None     None
41       None     None     None
```

(continues on next page)

(continued from previous page)

```
46      None    None    None
47      None    None    None
11      None    None    None
12      None    None    None
13      None    None    None
14      None    None    None
15      None    None    None
16      None    None    None
17      None    None    None
18      None    None    None
20      None    None    None
21      None    None    None
22      None    None    None
23      None    None    None
24      None    None    None
25      None    None    None
26      None    None    None
27      None    None    None
28      None    None    None
29      None    None    None
30      None    None    None
31      None    None    None
32      None    None    None
33      None    None    None
34      None    None    None
35      None    None    None
36      None    None    None
```

Analysis

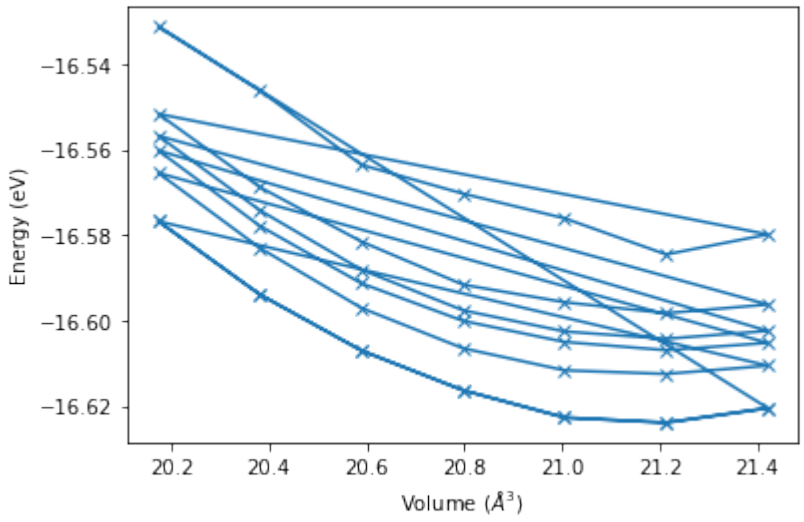
We aggregate the data for further processing in two separated lists, one for the volumes and one for the energies. To do so we iterate over the jobs within the project, filter the job names which contain the string 'vasp' and from those extract the final volume and the final energy.

[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.GpawJob, 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.

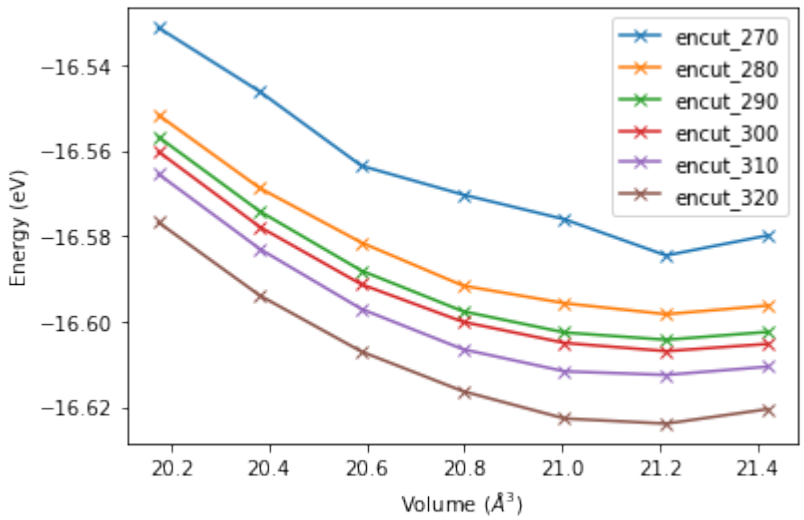
[10]:
```
for encut in np.linspace(270, 320, 6):
    encut_str = 'encut_' + str(int(encut))
    pr_encut = pr.open(encut_str)
    for strain in np.linspace(0.97, 1.03, 7):
        vasp_calculation_for_strain(pr=pr_encut,
                                    basis=basis,
                                    strain=strain,
                                    encut=encut)
```

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 0x2b8836c94470>
```



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]:
```
[-10938046483227.81, -160.21766207685127, 4.0, -21876092966476.383]
```

Vinet

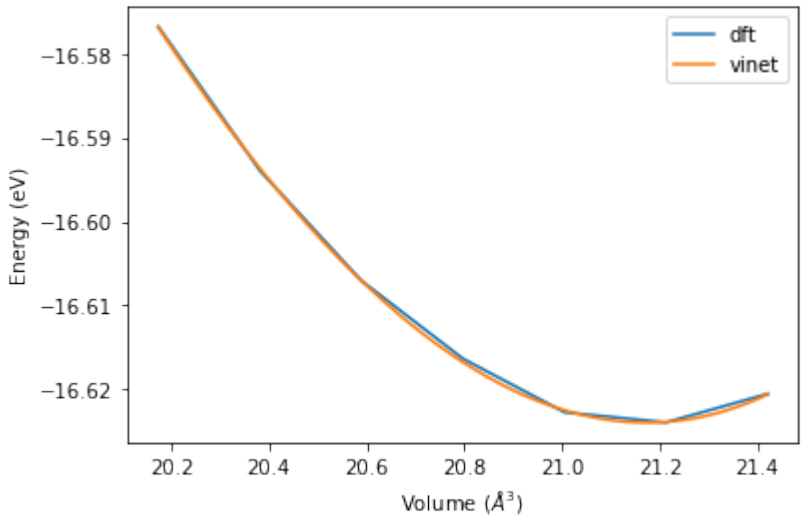
[14]:
```
[e0, b0, bP, v0], [e0_error, b0_error, bP_error, v0_error] = murn._fit_leastsq(volume_
↪lst=volume_lst,
                                                                                energy_
↪lst=energy_lst,
                                                                                ␣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

[14]:
```
[-16.62393775939236,
 361.18208413366904,
 -8.873197550150648,
 21.173041264923626]
```

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 0x2b8836cf24a8>
```



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.GpawJob, 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'] = 0.03
```

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

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

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

[20]:

	id	status	chemicalformula	job	subjob	projectpath	project
6	3601535	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
38	3601804	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
45	3602090	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
8	3602359	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
39	3602595	finished	None	gpaw_1_01	/gpaw_1_01	/cmmc/u/	
0	3602869	finished	None	gpaw_1_02	/gpaw_1_02	/cmmc/u/	
4	3603146	finished	None	gpaw_1_03	/gpaw_1_03	/cmmc/u/	
37	3603487	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	
5	3603765	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	
44	3604021	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	
2	3604219	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	
7	3604443	finished	None	gpaw_1_01	/gpaw_1_01	/cmmc/u/	
3	3604655	finished	None	gpaw_1_02	/gpaw_1_02	/cmmc/u/	
1	3604768	finished	None	gpaw_1_03	/gpaw_1_03	/cmmc/u/	
40	3604824	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	
42	3604871	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	
43	3604875	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	
48	3604880	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	
9	3604884	finished	None	gpaw_1_01	/gpaw_1_01	/cmmc/u/	
10	3604887	finished	None	gpaw_1_02	/gpaw_1_02	/cmmc/u/	
19	3604890	finished	None	gpaw_1_03	/gpaw_1_03	/cmmc/u/	
41	3604896	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	
46	3604899	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	
47	3604903	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	
11	3604907	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	
12	3604910	finished	None	gpaw_1_01	/gpaw_1_01	/cmmc/u/	
13	3604911	finished	None	gpaw_1_02	/gpaw_1_02	/cmmc/u/	
14	3604913	finished	None	gpaw_1_03	/gpaw_1_03	/cmmc/u/	
15	3604914	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	
16	3604915	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	
17	3604916	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	
18	3604917	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	
20	3604918	finished	None	gpaw_1_01	/gpaw_1_01	/cmmc/u/	
21	3604919	finished	None	gpaw_1_02	/gpaw_1_02	/cmmc/u/	
22	3604920	finished	None	gpaw_1_03	/gpaw_1_03	/cmmc/u/	
23	3604921	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	
24	3604922	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	
25	3604923	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	
26	3604924	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	
27	3604925	finished	None	gpaw_1_01	/gpaw_1_01	/cmmc/u/	
28	3604926	finished	None	gpaw_1_02	/gpaw_1_02	/cmmc/u/	
29	3604927	finished	None	gpaw_1_03	/gpaw_1_03	/cmmc/u/	
30	3604928	finished	None	gpaw_0_97	/gpaw_0_97	/cmmc/u/	
31	3604929	finished	None	gpaw_0_98	/gpaw_0_98	/cmmc/u/	
32	3604930	finished	None	gpaw_0_99	/gpaw_0_99	/cmmc/u/	
33	3604931	finished	None	gpaw_1_0	/gpaw_1_0	/cmmc/u/	
34	3604933	finished	None	gpaw_1_01	/gpaw_1_01	/cmmc/u/	
35	3604934	finished	None	gpaw_1_02	/gpaw_1_02	/cmmc/u/	
36	3604935	finished	None	gpaw_1_03	/gpaw_1_03	/cmmc/u/	

(continues on next page)

(continued from previous page)

```
39            janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
0             janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
4             janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/
37  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
5   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
44  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
2   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
7   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
3   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
1   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_270/
40  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
42  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
43  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
48  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
9   janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
10  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
19  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_280/
41  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
46  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
47  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
11  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
12  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
13  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
14  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_290/
15  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
16  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
17  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
18  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
20  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
21  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
22  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_300/
23  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
24  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
25  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
26  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
27  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
28  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
29  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_310/
30  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
31  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
32  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
33  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
34  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
35  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/
36  janj/pyiron/projects/2019/2019-09-04-website-examples/thermo/encut_320/

                    timestart timestop totalcputime        computer hamilton  \
6  2019-09-04 13:50:31.285688     None         None  janj@cmmc001#1  GpawJob
38 2019-09-04 13:50:42.970024     None         None  janj@cmmc001#1  GpawJob
45 2019-09-04 13:50:52.350144     None         None  janj@cmmc001#1  GpawJob
8  2019-09-04 13:51:01.413156     None         None  janj@cmmc001#1  GpawJob
39 2019-09-04 13:51:10.251511     None         None  janj@cmmc001#1  GpawJob
0  2019-09-04 13:51:20.041600     None         None  janj@cmmc001#1  GpawJob
4  2019-09-04 13:51:29.741836     None         None  janj@cmmc001#1  GpawJob
37 2019-09-04 13:51:42.796905     None         None  janj@cmmc001#1  GpawJob
5  2019-09-04 13:51:53.088836     None         None  janj@cmmc001#1  GpawJob
44 2019-09-04 13:52:02.979283     None         None  janj@cmmc001#1  GpawJob
```

(continues on next page)

(continued from previous page)

```
2  2019-09-04 13:52:11.921097     None         None  janj@cmmc001#1  GpawJob
7  2019-09-04 13:52:21.335934     None         None  janj@cmmc001#1  GpawJob
3  2019-09-04 13:52:31.009130     None         None  janj@cmmc001#1  GpawJob
1  2019-09-04 13:52:40.320534     None         None  janj@cmmc001#1  GpawJob
40 2019-09-04 13:52:50.470563     None         None  janj@cmmc001#1  GpawJob
42 2019-09-04 13:52:59.954797     None         None  janj@cmmc001#1  GpawJob
43 2019-09-04 13:53:09.963464     None         None  janj@cmmc001#1  GpawJob
48 2019-09-04 13:53:20.141701     None         None  janj@cmmc001#1  GpawJob
9  2019-09-04 13:53:29.324131     None         None  janj@cmmc001#1  GpawJob
10 2019-09-04 13:53:38.550641     None         None  janj@cmmc001#1  GpawJob
19 2019-09-04 13:53:48.099532     None         None  janj@cmmc001#1  GpawJob
41 2019-09-04 13:53:59.152323     None         None  janj@cmmc001#1  GpawJob
46 2019-09-04 13:54:08.057464     None         None  janj@cmmc001#1  GpawJob
47 2019-09-04 13:54:17.516512     None         None  janj@cmmc001#1  GpawJob
11 2019-09-04 13:54:26.874849     None         None  janj@cmmc001#1  GpawJob
12 2019-09-04 13:54:37.345194     None         None  janj@cmmc001#1  GpawJob
13 2019-09-04 13:54:48.785761     None         None  janj@cmmc001#1  GpawJob
14 2019-09-04 13:55:00.556380     None         None  janj@cmmc001#1  GpawJob
15 2019-09-04 13:55:10.828970     None         None  janj@cmmc001#1  GpawJob
16 2019-09-04 13:55:19.451476     None         None  janj@cmmc001#1  GpawJob
17 2019-09-04 13:55:28.235999     None         None  janj@cmmc001#1  GpawJob
18 2019-09-04 13:55:36.912405     None         None  janj@cmmc001#1  GpawJob
20 2019-09-04 13:55:46.777440     None         None  janj@cmmc001#1  GpawJob
21 2019-09-04 13:55:55.186420     None         None  janj@cmmc001#1  GpawJob
22 2019-09-04 13:56:05.185718     None         None  janj@cmmc001#1  GpawJob
23 2019-09-04 13:56:14.138209     None         None  janj@cmmc001#1  GpawJob
24 2019-09-04 13:56:22.072544     None         None  janj@cmmc001#1  GpawJob
25 2019-09-04 13:56:30.773140     None         None  janj@cmmc001#1  GpawJob
26 2019-09-04 13:56:38.738514     None         None  janj@cmmc001#1  GpawJob
27 2019-09-04 13:56:46.853680     None         None  janj@cmmc001#1  GpawJob
28 2019-09-04 13:56:55.404483     None         None  janj@cmmc001#1  GpawJob
29 2019-09-04 13:57:03.940368     None         None  janj@cmmc001#1  GpawJob
30 2019-09-04 13:57:13.347542     None         None  janj@cmmc001#1  GpawJob
31 2019-09-04 13:57:21.459612     None         None  janj@cmmc001#1  GpawJob
32 2019-09-04 13:57:29.623085     None         None  janj@cmmc001#1  GpawJob
33 2019-09-04 13:57:37.903535     None         None  janj@cmmc001#1  GpawJob
34 2019-09-04 13:57:46.032614     None         None  janj@cmmc001#1  GpawJob
35 2019-09-04 13:57:54.280901     None         None  janj@cmmc001#1  GpawJob
36 2019-09-04 13:58:02.411374     None         None  janj@cmmc001#1  GpawJob

   hamversion parentid masterid
6        None     None     None
38       None     None     None
45       None     None     None
8        None     None     None
39       None     None     None
0        None     None     None
4        None     None     None
37       None     None     None
5        None     None     None
44       None     None     None
2        None     None     None
7        None     None     None
3        None     None     None
1        None     None     None
40       None     None     None
42       None     None     None
```

(continues on next page)

(continued from previous page)

```
43       None     None     None
48       None     None     None
9        None     None     None
10       None     None     None
19       None     None     None
41       None     None     None
46       None     None     None
47       None     None     None
11       None     None     None
12       None     None     None
13       None     None     None
14       None     None     None
15       None     None     None
16       None     None     None
17       None     None     None
18       None     None     None
20       None     None     None
21       None     None     None
22       None     None     None
23       None     None     None
24       None     None     None
25       None     None     None
26       None     None     None
27       None     None     None
28       None     None     None
29       None     None     None
30       None     None     None
31       None     None     None
32       None     None     None
33       None     None     None
34       None     None     None
35       None     None     None
36       None     None     None
```

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

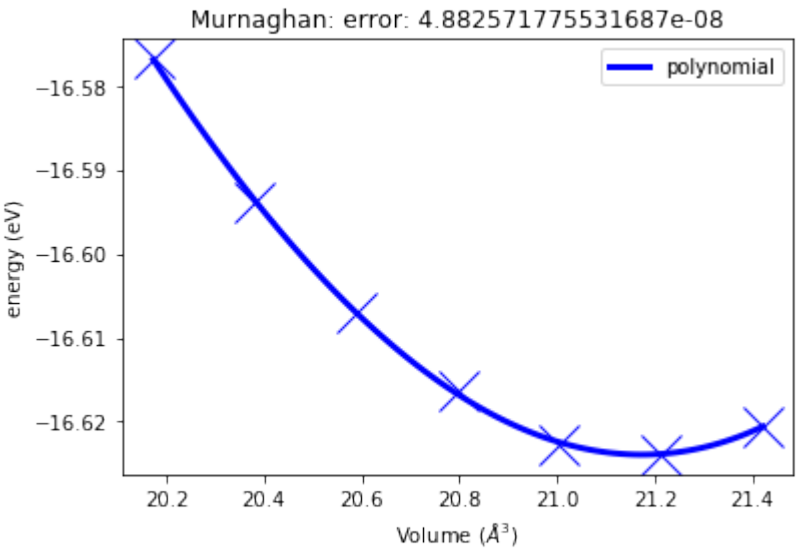
```
The job murn was saved and received the ID: 3606074
The job strain_0_97 was saved and received the ID: 3606075
The job strain_0_98 was saved and received the ID: 3606084
The job strain_0_99 was saved and received the ID: 3606089
The job strain_1_0 was saved and received the ID: 3606092
The job strain_1_01 was saved and received the ID: 3606099
The job strain_1_02 was saved and received the ID: 3606106
The job strain_1_03 was saved and received the ID: 3606112
job_id:  3606075 finished
job_id:  3606084 finished
job_id:  3606089 finished
job_id:  3606092 finished
job_id:  3606099 finished
job_id:  3606106 finished
job_id:  3606112 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  20.172969 -16.576797    0.0  3606075            -8.102283
1  20.380937 -16.593942    0.0  3606084            -8.102283
2  20.588906 -16.607049    0.0  3606089            -8.102283
3  20.796875 -16.616336    0.0  3606092            -8.102283
4  21.004844 -16.622714    0.0  3606099            -8.102283
5  21.212813 -16.623909    0.0  3606106            -8.102283
6  21.420781 -16.620513    0.0  3606112            -8.102283

   equilibrium_bulk_modulus  equilibrium_energy  equilibrium_volume
0                359.180621          -16.623924           21.172823
1                359.180621          -16.623924           21.172823
2                359.180621          -16.623924           21.172823
3                359.180621          -16.623924           21.172823
4                359.180621          -16.623924           21.172823
5                359.180621          -16.623924           21.172823
6                359.180621          -16.623924           21.172823
```

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



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

[24]:
```
{'fit_type': 'vinet',
 'volume_eq': 21.173041264923626,
 'energy_eq': -16.62393775939236,
 'bulkmodul_eq': 361.18208413366904,
 'b_prime_eq': -8.873197550150648,
 'least_square_error': array([2.35206844e-04, 2.03161020e+01, 3.15362473e+00, 7.
↪39621643e-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(0.97, 1.03, 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(0.97, 1.03, 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]:
```
([20.17296874999999,
  20.380937499999995,
  20.588906250000004,
  20.796874999999996,
  21.004843749999992,
  21.212812500000016,
  21.42078124999999],
 [20.17296874999999,
  19.769509374999995,
  19.571814281250003,
  19.571814281250003,
  19.76753242406251,
  20.162883072543767,
  20.767769564720073])
```

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()
```

(continues on next page)

(continued from previous page)

```
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()
```

[ ]:

4.3.3 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.project 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()
```

Using the ASE spacegroup class

[7]:
```
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)
```

[8]:
```
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

[9]:
```
structure = pr.create_ase_bulk('Al', cubic=True)
```

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

[10]:
```
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:
[[ 4.05  0.    0.  ]
 [ 0.    4.05  0.  ]
 [ 0.    0.    4.05]]
```

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

[11]:
```
structure.cell
```

[11]:
```
array([[ 4.05,  0.  ,  0.  ],
       [ 0.  ,  4.05,  0.  ],
       [ 0.  ,  0.  ,  4.05]])
```

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

[12]:
```
structure.positions
```

[12]:
```
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

[13]:
```
structure = pr.create_ase_bulk('Al', cubic=True)
structure.set_repeat([4,4,4])
```

To create the vacancy at position index “0” simply use:

[14]:
```
del structure[0]
```

To plot the structure that now contains a vacancy run:

[15]:
```
structure.plot3d()
```

```
NGLWidget()
```

Creating multiple vacancies

[16]:
```
# 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

[17]:
```
# 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]
```

[18]:
```
# 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

[19]:
```
# 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

[20]:
```
n_sub = 24
structure[np.random.permutation(len(structure))[:n_sub]] = 'Mg'
```

[21]:
```
# 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

[22]:
```
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()
```

Surfaces (with ASE)

Some more commonly studied surfaces can also be defined with the `create_surface()` function based on the ASE surface builder

[23]:
```
fcc111 = pr.create_surface("Pt",
                           surface_type="fcc111",
                           size=(3, 4, 5),
                           a=4.2,
                           vacuum=20)
```

[24]:
```
fcc111.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)

[25]:
```
# The COD structures can be accessed through their unique COD identifier
filename = '1008753.cif'
url = 'http://www.crystallography.net/cod/{}'.format(filename)
```

[26]:
```
# Download and save the structure file locally
import urllib
urllib.request.urlretrieve(url=url, filename='strucs.'+filename);
```

[27]:
```
# 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'))
```

[28]:
```
structure.plot3d()
```

```
NGLWidget()
```

[ ]:
```

```

4.3.4 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.project import Project
import numpy as np
```

(continues on next page)

(continued from previous page)

```
%matplotlib  inline
import matplotlib.pylab as plt
import seaborn as sns
```

[2]:
```
pr = Project("PhonopyExample")
pot =  'Al_Mg_Mendelev_eam'
pr.remove_jobs(recursive=True)
```

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

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

[4]:
```
def scale_structure(struct, scale):
    """
    Rescale the atomic positions and cell of a structure simultaneously.
    Accepts rescaling by an arbitrary real-valued 3x3 numpy array, but a float can be␣
↪given
    for isotropic rescaling.

    Args:
        struct (Structure object): The structure to rescale.
        scale (float or np.array(3,3)): The matrix to rescale by. (float -> isotropic.
↪)

    Returns:
        A rescaled copy of the structure.

    ..TODO: Double check that the scaling matrix still spans 3-space (determinant␣
↪check?)
```

(continues on next page)

(continued from previous page)

```
    """
    if isinstance(scale, float) or isinstance(scale, int):
        scale_mat = scale * np.eye(3)
    else:
        assert(scale.shape == (3,3))
        scale_mat = scale.T
    new_struct = struct.copy()
    new_struct.cell = np.dot(struct.cell, scale_mat)
    new_struct.positions = np.dot(struct.positions, scale_mat)
    return new_struct
```

[5]:
```
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?

[6]:
```
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.

[7]:
```
job_unit = pr_te.create_job(pr.job_type.Lammps, "UnitCell")
```

[8]:
```
basis = pr_te.create_structure("Al", "fcc", 4.04)
```

[9]:
```
job_unit.structure = basis
job_unit.potential = pot
```

[10]:
```
job_unit.calc_minimize(pressure=0.0)
job_unit.run()
```

```
The job UnitCell was saved and received the ID: 3596380
```

[11]:
```
basis_rel = job_unit.get_final_structure()
```

Relax the bulk supercell

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

[12]:
```
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)
```

[13]:
```
n_reps = 3
job_bulk_1.structure = basis_rel.repeat(rep=n_reps)
job_bulk_1.potential = pot
```

[14]:
```
job_bulk_1.structure.plot3d();
```

```
_ColormakerRegistry()
```

[15]:
```
job_bulk_1.calc_minimize(pressure=0.0)
job_bulk_1.run()
```

```
The job Bulk_1 was saved and received the ID: 3596381
```

Calculate phonons

Run phonopy on the bulk supercell

[16]:
```
phono_bulk_1 = make_phonopy_job(job_bulk_1, "PhonoBulk_1")
```

[17]:
```
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: 3596382
The job supercell_phonon_0 was saved and received the ID: 3596383
```

[18]:
```
# 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
```

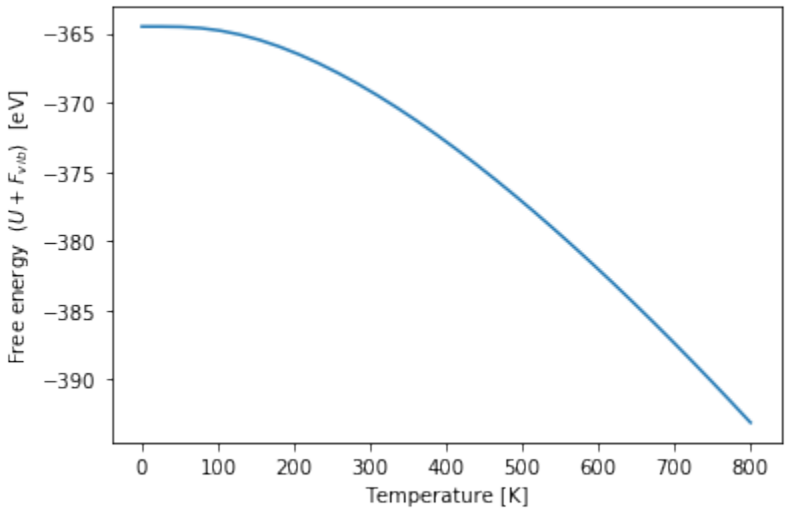
[19]:
```
U_bulk_1 = job_bulk_1.output.energy_pot[-1]
Fvib_bulk_1 = tp_bulk_1.free_energies
plt.plot(temperatures, U_bulk_1 + Fvib_bulk_1)
```

(continues on next page)

(continued from previous page)

```
plt.xlabel("Temperature [K]")
plt.ylabel("Free energy  ($U+F_{vib}$)  [eV]")
```

[19]:
```
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.

[20]:
```
# According to Wikipedia, the thermal expansivity is about 0.0023% / Kelvin
# So at our maximum temperature, we expect around 1.8% expansion
scale_min = 0.995
scale_max = 1.02
scale_step = 0.002
scales = np.linspace(scale_min, scale_max, int((scale_max - scale_min) / scale_step))
```

[21]:
```
# Let's keep things clean by making another sub-directory
pr_scales = pr_te.create_group("ScanScales")
```

[22]:
```
# 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(".", "_"))

    # 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 = scale_structure(sc_bulk_rel, scale)
    job_bulk.calc_static()
    job_bulk.run()
    U = job_bulk.output.energy_tot[-1]
```

(continues on next page)

(continued from previous page)

```
    # 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)
    Fvib = tp_bulk.free_energies

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

```
The job Bulk_0_995 was saved and received the ID: 3596385
The job PhonoBulk_0_995 was saved and received the ID: 3596386
The job supercell_phonon_0 was saved and received the ID: 3596387
The job Bulk_0_9972727272727273 was saved and received the ID: 3596388
The job PhonoBulk_0_9972727272727273 was saved and received the ID: 3596389
The job supercell_phonon_0 was saved and received the ID: 3596390
The job Bulk_0_9995454545454545 was saved and received the ID: 3596394
The job PhonoBulk_0_9995454545454545 was saved and received the ID: 3596400
The job supercell_phonon_0 was saved and received the ID: 3596405
The job Bulk_1_0018181818181817 was saved and received the ID: 3596418
The job PhonoBulk_1_0018181818181817 was saved and received the ID: 3596419
The job supercell_phonon_0 was saved and received the ID: 3596420
The job Bulk_1_0040909090909091 was saved and received the ID: 3596434
The job PhonoBulk_1_0040909090909091 was saved and received the ID: 3596436
The job supercell_phonon_0 was saved and received the ID: 3596439
The job Bulk_1_0063636363636363 was saved and received the ID: 3596449
The job PhonoBulk_1_0063636363636363 was saved and received the ID: 3596450
The job supercell_phonon_0 was saved and received the ID: 3596451
The job Bulk_1_0086363636363636 was saved and received the ID: 3596456
The job PhonoBulk_1_0086363636363636 was saved and received the ID: 3596458
The job supercell_phonon_0 was saved and received the ID: 3596461
The job Bulk_1_010909090909091 was saved and received the ID: 3596473
The job PhonoBulk_1_010909090909091 was saved and received the ID: 3596475
The job supercell_phonon_0 was saved and received the ID: 3596478
The job Bulk_1_0131818181818182 was saved and received the ID: 3596488
The job PhonoBulk_1_0131818181818182 was saved and received the ID: 3596490
The job supercell_phonon_0 was saved and received the ID: 3596494
The job Bulk_1_0154545454545454 was saved and received the ID: 3596505
The job PhonoBulk_1_0154545454545454 was saved and received the ID: 3596507
The job supercell_phonon_0 was saved and received the ID: 3596510
The job Bulk_1_0177272727272728 was saved and received the ID: 3596521
The job PhonoBulk_1_0177272727272728 was saved and received the ID: 3596523
The job supercell_phonon_0 was saved and received the ID: 3596526
The job Bulk_1_02 was saved and received the ID: 3596537
The job PhonoBulk_1_02 was saved and received the ID: 3596539
The job supercell_phonon_0 was saved and received the ID: 3596542
```

[23]:
```
# The lattice constant is probably a more informative value than the 0K-relative␣
↪strain
latts = basis_rel.cell[0][0] * scales
```

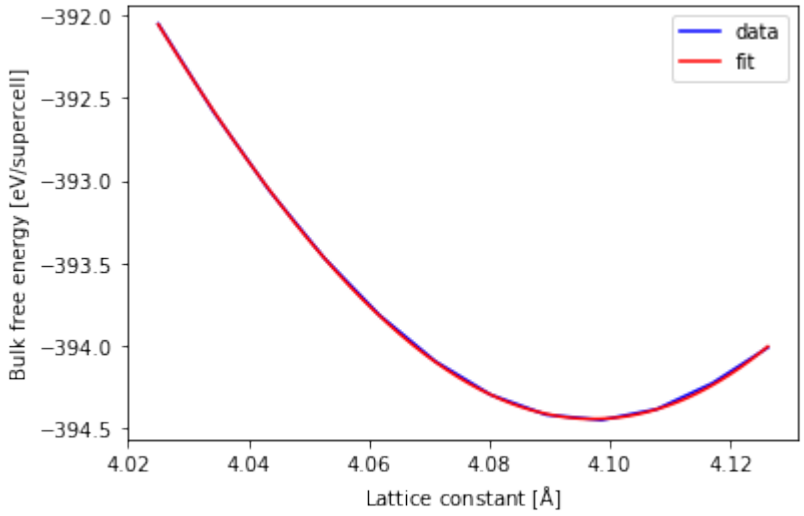
[24]:
```
# At each temperature, find the optimal volume by a simple quadratic fit
# ...Wait, which order fit will be good enough? Let's just spot-check
free_en = bulk_free_energies[:, -1]
plt.plot(latts, free_en, color='b', label='data')
```

(continues on next page)

(continued from previous page)

```
# 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)
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
```

[24]:
```
<matplotlib.legend.Legend at 0x2b61a42bf278>
```



[25]:
```
# 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))]
```

[26]:
```
# Check that they're resonable
print(best_latt_guess, '\n', best_latts)
```

```
4.045270475668763
 [4.05946291 4.05949371 4.05987201 4.06083718 4.06226186 4.06396024
 4.06579637 4.06768361 4.06956868 4.0714193  4.07321635 4.07494901
 4.07661182 4.07820271 4.07972185 4.08117076 4.08255184 4.08386799
 4.08512237 4.08631821 4.08745877 4.0885472  4.08958656 4.09057975
 4.0915295  4.09243842 4.09330892 4.09414326 4.09494357 4.09571182
 4.09644984 4.09715935]
```

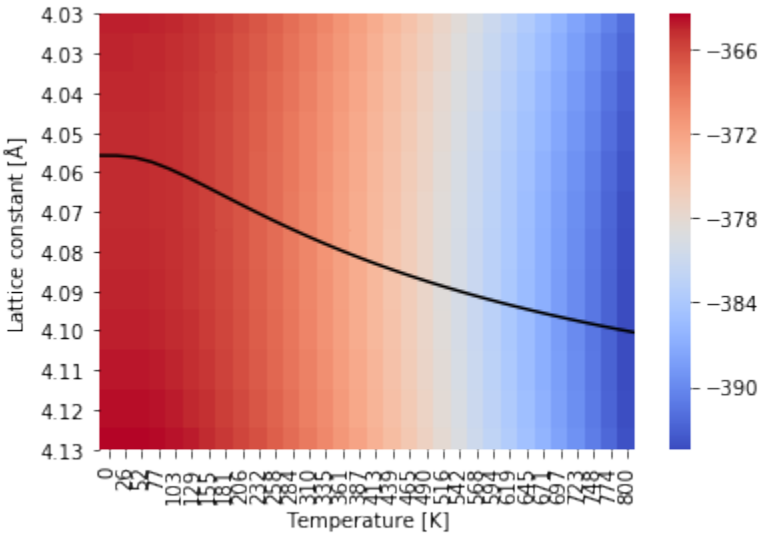
[27]:
```
# Let's look at the landscape
fig, ax = plt.subplots()
sns.heatmap(bulk_free_energies, ax=ax, cmap="coolwarm",
            xticklabels=['{:,.0f}'.format(T) for T in temperatures],
            yticklabels=['{:,.2f}'.format(a) for a in latts])
```

(continues on next page)

(continued from previous page)

```
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


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

[27]:
```
[<matplotlib.lines.Line2D at 0x2b61d9e8ee80>]
```



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

[28]:
```
pr_vac = pr.create_group("Vacancies")
```

[29]:
```
# 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.]
[2.02263524 2.02263524 0.        ]
```

[30]:
```
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
```

[31]:
```
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
```

[32]:
```
# 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: 3596547
The job VacancyFinal was saved and received the ID: 3596549
```

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.

[33]:
```
phon_vac_i = make_phonopy_job(job_vac_i, "PhonoVacInitial")
phon_vac_f = make_phonopy_job(job_vac_f, "PhonoVacFinal")
```

[34]:
```
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: 3596552
The job supercell_phonon_0 was saved and received the ID: 3596554
The job supercell_phonon_1 was saved and received the ID: 3596556
The job supercell_phonon_2 was saved and received the ID: 3596559
The job supercell_phonon_3 was saved and received the ID: 3596561
The job supercell_phonon_4 was saved and received the ID: 3596564
```

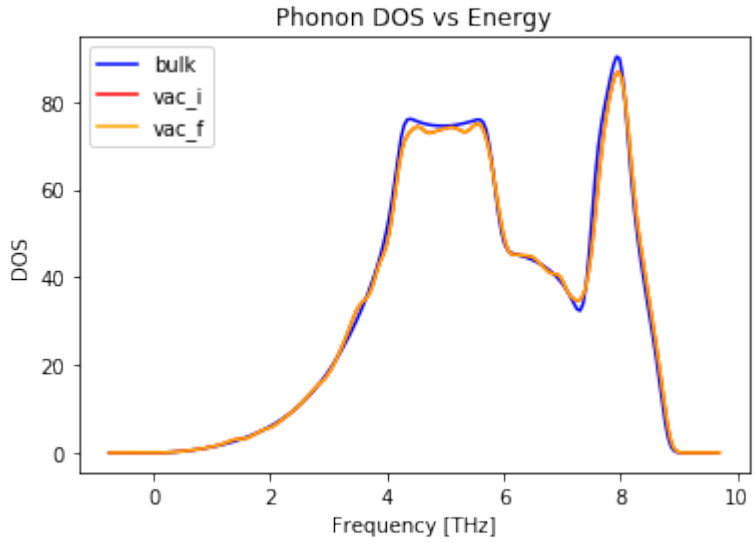
(continues on next page)

(continued from previous page)

```
The job supercell_phonon_5 was saved and received the ID: 3596566
The job supercell_phonon_6 was saved and received the ID: 3596569
The job supercell_phonon_7 was saved and received the ID: 3596571
The job supercell_phonon_8 was saved and received the ID: 3596573
The job supercell_phonon_9 was saved and received the ID: 3596576
The job supercell_phonon_10 was saved and received the ID: 3596578
The job supercell_phonon_11 was saved and received the ID: 3596580
The job supercell_phonon_12 was saved and received the ID: 3596582
The job supercell_phonon_13 was saved and received the ID: 3596585
The job supercell_phonon_14 was saved and received the ID: 3596587
The job supercell_phonon_15 was saved and received the ID: 3596589
The job supercell_phonon_16 was saved and received the ID: 3596592
The job supercell_phonon_17 was saved and received the ID: 3596594
The job supercell_phonon_18 was saved and received the ID: 3596597
The job supercell_phonon_19 was saved and received the ID: 3596599
The job supercell_phonon_20 was saved and received the ID: 3596601
The job PhonoVacFinal was saved and received the ID: 3596613
The job supercell_phonon_0 was saved and received the ID: 3596616
The job supercell_phonon_1 was saved and received the ID: 3596618
The job supercell_phonon_2 was saved and received the ID: 3596620
The job supercell_phonon_3 was saved and received the ID: 3596623
The job supercell_phonon_4 was saved and received the ID: 3596625
The job supercell_phonon_5 was saved and received the ID: 3596628
The job supercell_phonon_6 was saved and received the ID: 3596630
The job supercell_phonon_7 was saved and received the ID: 3596632
The job supercell_phonon_8 was saved and received the ID: 3596635
The job supercell_phonon_9 was saved and received the ID: 3596637
The job supercell_phonon_10 was saved and received the ID: 3596640
The job supercell_phonon_11 was saved and received the ID: 3596642
The job supercell_phonon_12 was saved and received the ID: 3596644
The job supercell_phonon_13 was saved and received the ID: 3596646
The job supercell_phonon_14 was saved and received the ID: 3596649
The job supercell_phonon_15 was saved and received the ID: 3596651
The job supercell_phonon_16 was saved and received the ID: 3596653
The job supercell_phonon_17 was saved and received the ID: 3596655
The job supercell_phonon_18 was saved and received the ID: 3596658
The job supercell_phonon_19 was saved and received the ID: 3596659
The job supercell_phonon_20 was saved and received the ID: 3596660
```

[35]:
```
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()
```

[35]:
```
<matplotlib.legend.Legend at 0x2b61da20bcc0>
```



Attack frequency

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

[36]:
```
# 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)
```

[37]:
```
job_vac_ts = pr_vac.create_job(pr.job_type.Lammps, "VacancyTransition")
job_vac_ts.potential = pot
job_vac_ts.structure = sc_vac_ts
```

[38]:
```
# 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: 3596670
```

[39]:
```
phon_vac_ts = make_phonopy_job(job_vac_ts, "PhonoVacTransition")
```

[40]:
```
phon_vac_ts.run()
tp_vac_ts = phon_vac_ts.get_thermal_properties(temperatures=temperatures)
```

```
The job PhonoVacTransition was saved and received the ID: 3596673
The job supercell_phonon_0 was saved and received the ID: 3596676
The job supercell_phonon_1 was saved and received the ID: 3596678
The job supercell_phonon_2 was saved and received the ID: 3596681
The job supercell_phonon_3 was saved and received the ID: 3596683
The job supercell_phonon_4 was saved and received the ID: 3596686
The job supercell_phonon_5 was saved and received the ID: 3596688
The job supercell_phonon_6 was saved and received the ID: 3596691
The job supercell_phonon_7 was saved and received the ID: 3596693
The job supercell_phonon_8 was saved and received the ID: 3596695
The job supercell_phonon_9 was saved and received the ID: 3596698
```

(continues on next page)

(continued from previous page)

```
The job supercell_phonon_10 was saved and received the ID: 3596700
The job supercell_phonon_11 was saved and received the ID: 3596703
The job supercell_phonon_12 was saved and received the ID: 3596705
The job supercell_phonon_13 was saved and received the ID: 3596707
The job supercell_phonon_14 was saved and received the ID: 3596709
The job supercell_phonon_15 was saved and received the ID: 3596712
The job supercell_phonon_16 was saved and received the ID: 3596714
The job supercell_phonon_17 was saved and received the ID: 3596716
The job supercell_phonon_18 was saved and received the ID: 3596719
The job supercell_phonon_19 was saved and received the ID: 3596721
The job supercell_phonon_20 was saved and received the ID: 3596723
The job supercell_phonon_21 was saved and received the ID: 3596726
The job supercell_phonon_22 was saved and received the ID: 3596728
The job supercell_phonon_23 was saved and received the ID: 3596731
The job supercell_phonon_24 was saved and received the ID: 3596733
The job supercell_phonon_25 was saved and received the ID: 3596735
The job supercell_phonon_26 was saved and received the ID: 3596738
The job supercell_phonon_27 was saved and received the ID: 3596740
The job supercell_phonon_28 was saved and received the ID: 3596742
The job supercell_phonon_29 was saved and received the ID: 3596744
The job supercell_phonon_30 was saved and received the ID: 3596746
The job supercell_phonon_31 was saved and received the ID: 3596749
The job supercell_phonon_32 was saved and received the ID: 3596751
The job supercell_phonon_33 was saved and received the ID: 3596754
The job supercell_phonon_34 was saved and received the ID: 3596756
The job supercell_phonon_35 was saved and received the ID: 3596758
The job supercell_phonon_36 was saved and received the ID: 3596760
The job supercell_phonon_37 was saved and received the ID: 3596762
The job supercell_phonon_38 was saved and received the ID: 3596765
The job supercell_phonon_39 was saved and received the ID: 3596767
The job supercell_phonon_40 was saved and received the ID: 3596769
The job supercell_phonon_41 was saved and received the ID: 3596772
The job supercell_phonon_42 was saved and received the ID: 3596774
The job supercell_phonon_43 was saved and received the ID: 3596776
The job supercell_phonon_44 was saved and received the ID: 3596778
The job supercell_phonon_45 was saved and received the ID: 3596781
The job supercell_phonon_46 was saved and received the ID: 3596782
The job supercell_phonon_47 was saved and received the ID: 3596783
The job supercell_phonon_48 was saved and received the ID: 3596784
The job supercell_phonon_49 was saved and received the ID: 3596785
The job supercell_phonon_50 was saved and received the ID: 3596786
The job supercell_phonon_51 was saved and received the ID: 3596788
The job supercell_phonon_52 was saved and received the ID: 3596790
The job supercell_phonon_53 was saved and received the ID: 3596792
The job supercell_phonon_54 was saved and received the ID: 3596794
The job supercell_phonon_55 was saved and received the ID: 3596796
The job supercell_phonon_56 was saved and received the ID: 3596798
The job supercell_phonon_57 was saved and received the ID: 3596799
The job supercell_phonon_58 was saved and received the ID: 3596801
The job supercell_phonon_59 was saved and received the ID: 3596803
The job supercell_phonon_60 was saved and received the ID: 3596806
The job supercell_phonon_61 was saved and received the ID: 3596808
The job supercell_phonon_62 was saved and received the ID: 3596810
The job supercell_phonon_63 was saved and received the ID: 3596813
The job supercell_phonon_64 was saved and received the ID: 3596815
The job supercell_phonon_65 was saved and received the ID: 3596817
The job supercell_phonon_66 was saved and received the ID: 3596819
```

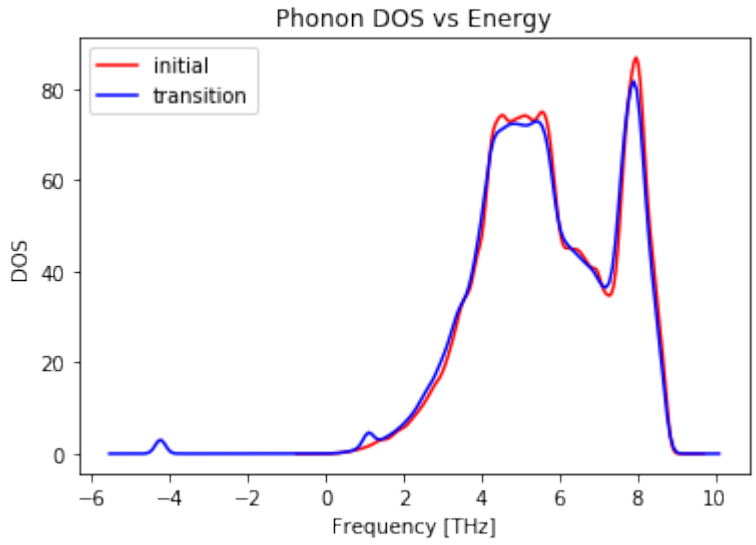
(continues on next page)

(continued from previous page)

```
The job supercell_phonon_67 was saved and received the ID: 3596822
The job supercell_phonon_68 was saved and received the ID: 3596824
The job supercell_phonon_69 was saved and received the ID: 3596826
The job supercell_phonon_70 was saved and received the ID: 3596829
The job supercell_phonon_71 was saved and received the ID: 3596831
The job supercell_phonon_72 was saved and received the ID: 3596833
The job supercell_phonon_73 was saved and received the ID: 3596836
The job supercell_phonon_74 was saved and received the ID: 3596838
The job supercell_phonon_75 was saved and received the ID: 3596842
The job supercell_phonon_76 was saved and received the ID: 3596846
The job supercell_phonon_77 was saved and received the ID: 3596850
The job supercell_phonon_78 was saved and received the ID: 3596855
The job supercell_phonon_79 was saved and received the ID: 3596858
The job supercell_phonon_80 was saved and received the ID: 3596863
The job supercell_phonon_81 was saved and received the ID: 3596866
The job supercell_phonon_82 was saved and received the ID: 3596869
The job supercell_phonon_83 was saved and received the ID: 3596874
The job supercell_phonon_84 was saved and received the ID: 3596878
The job supercell_phonon_85 was saved and received the ID: 3596881
The job supercell_phonon_86 was saved and received the ID: 3596885
The job supercell_phonon_87 was saved and received the ID: 3596890
The job supercell_phonon_88 was saved and received the ID: 3596894
The job supercell_phonon_89 was saved and received the ID: 3596898
The job supercell_phonon_90 was saved and received the ID: 3596902
The job supercell_phonon_91 was saved and received the ID: 3596907
The job supercell_phonon_92 was saved and received the ID: 3596911
The job supercell_phonon_93 was saved and received the ID: 3596915
The job supercell_phonon_94 was saved and received the ID: 3596920
```

[41]:
```
# 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()
```

[41]:
```
<matplotlib.legend.Legend at 0x2b61db5474e0>
```



To calculate the attack 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.

[42]:
```
freq_i = phon_vac_i.phonopy.get_frequencies(0)[3:]
freq_ts = phon_vac_i.phonopy.get_frequencies(0)[4:]
```

[43]:
```
print(np.prod(freq_i))
```

```
6.870675816849329e+236
```

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

[44]:
```
# Products are dangerous beasts, so we'll do a little numeric magic
nu = np.prod(freq_i[:-1] / freq_ts) * freq_i[-1]
print("Attack frequency is ", nu, "THz (10^-12 s)")
```

```
Attack frequency is  2.6826827779812032 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.

[ ]:

4.3.5 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.project 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,
```

(continues on next page)

(continued from previous page)

```
                    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',
 '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()
```

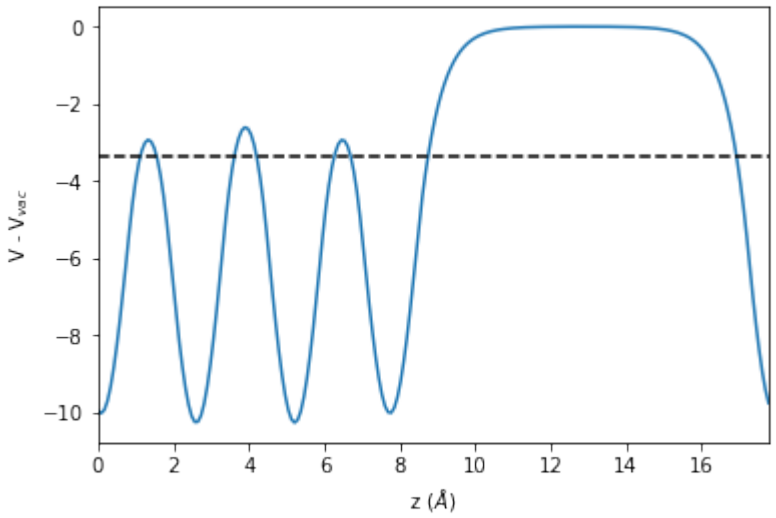
Post processing

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)
    #ham.server.cores = 20
    #ham.server.queue = queue
```

(continues on next page)

(continued from previous page)

```
    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
```

[ ]:
```
```

4.3.6 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.project import Project
import ase.units as units
```

[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)
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]:
```
job_name = "water_slow"
ham = pr.create_job("Lammps", job_name)
ham.structure = water
# Listing available potentials for this structure
ham.list_potentials()
```

[5]:
```
['H2O_tip3p', 'H2O_tip3p_slab']
```

We will use the `H2O_tip3p` potential. The `H2O_tip3p_slab` should be used if you want to switch off the periodic boundary conditions along the z-axis

[6]:
```
ham.potential = 'H2O_tip3p'
ham.calc_md(temperature=300,
            n_ionic_steps=1e4,
            time_step=0.01)
ham.run()
```

[7]:
```
view = ham.animate_structure()
view
```

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

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

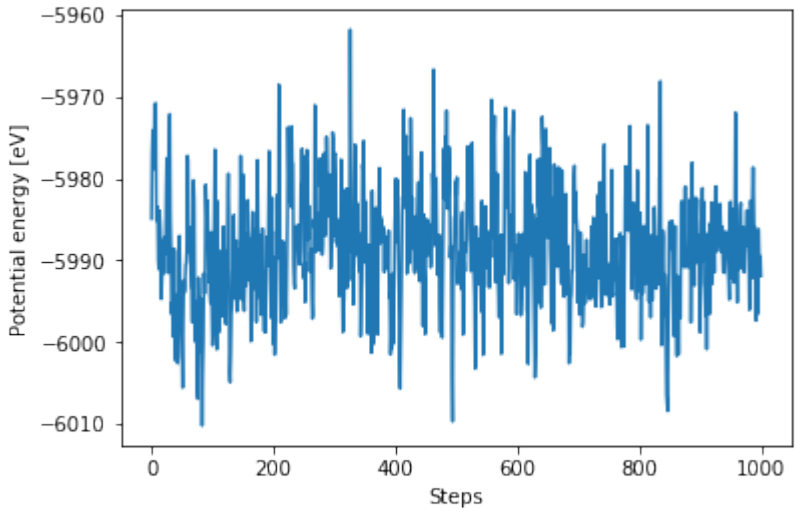
[8]:
```
# Get the final structure from the previous simulation
struct = ham.get_structure(iteration_step=-1)
job_name = "water_fast"
ham_eq = pr.create_job("Lammps", job_name)
ham_eq.structure = struct
ham_eq.potential = 'H2O_tip3p'
ham_eq.calc_md(temperature=300,
               n_ionic_steps=1e4,
               n_print=10,
               time_step=1)
ham_eq.run()
```

[9]:
```
view = ham_eq.animate_structure()
view
```

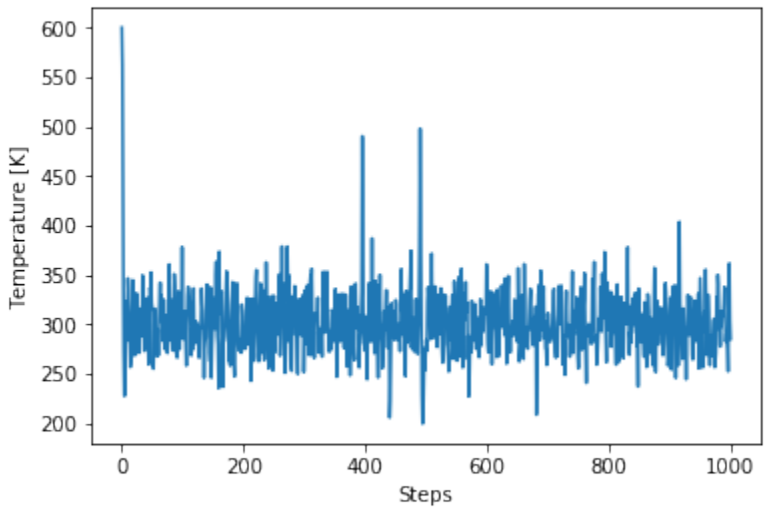
```
NGLWidget(count=1001)
```

We can now plot the trajectories

[10]:
```
plt.plot(ham_eq["output/generic/energy_pot"])
plt.xlabel("Steps")
plt.ylabel("Potential energy [eV]");
```



[11]:
```
plt.plot(ham_eq["output/generic/temperature"])
plt.xlabel("Steps")
plt.ylabel("Temperature [K]");
```



Structure analysis

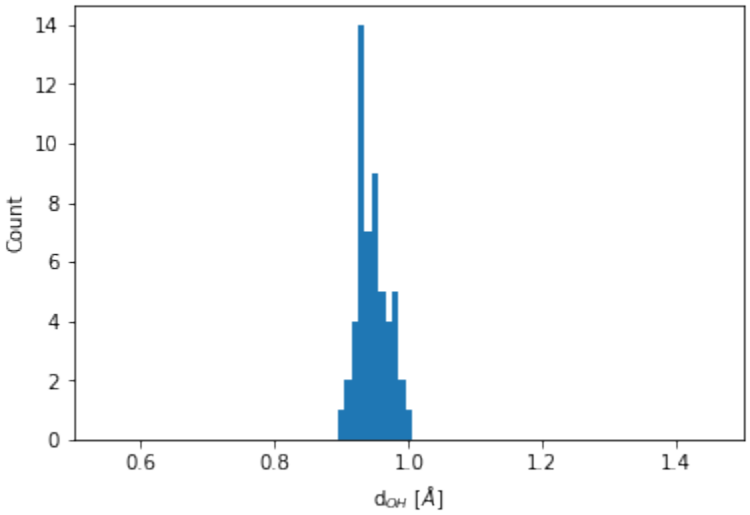
We will now use the `get_neighbors()` function to determine structural properties from the computed trajectories. We take advantage of the fact that the TIP3P water model is a rigid water model which means the neighbors of each molecule never change. Therefore they need to be indexed only once

[12]:
```
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:

[13]:
```
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("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 controlling the cutoff distance

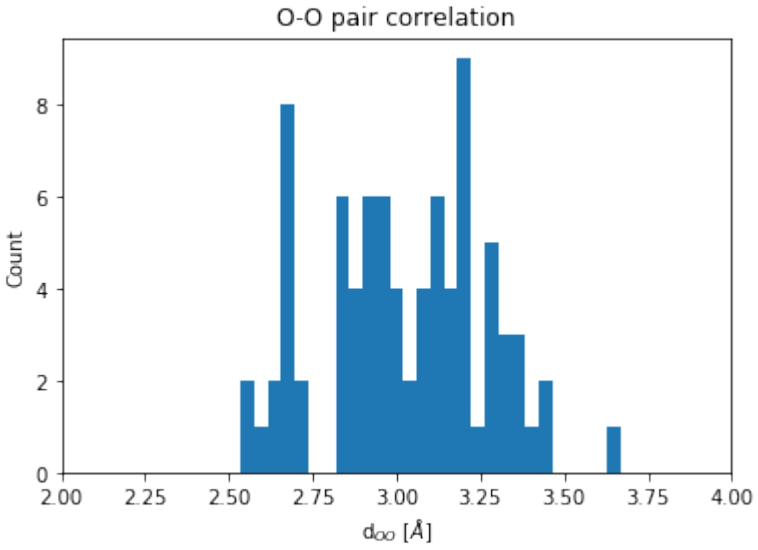
[14]:
```
neighbors = final_struct.get_neighbors(cutoff=8)
```

[15]:
```
neigh_indices = np.hstack(neighbors.indices[O_indices])
neigh_distances = np.hstack(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:

[16]:
```
# Getting the neighboring Oxyhen indices
O_neigh_indices  = np.in1d(neigh_indices, O_indices)
O_neigh_distances = neigh_distances[O_neigh_indices]
```

[17]:
```
bins = np.linspace(1, 5, 100)
count = plt.hist(O_neigh_distances, bins=bins)
plt.xlim(2, 4)
plt.title("O-O pair correlation")
plt.xlabel("d$_{OO}$ [$\AA$]")
plt.ylabel("Count");
```



[ ]:

4.3.7 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

---

4.4 Team

pyiron was initially developed in the Computational Materials Design department of Joerg Neugebauer at the Max Planck Insitut für Eisenforschung/ Max Planck Insitute for iron research (MPIE) as a framework for ab initio thermo dynamics. In collaboration with the Interdisciplinary Centre for Advanced Materials Simulation (ICAMS) the framework was recently extended for high throughput applications resulting in the opensource release of the pyiron.

4.4.1 Core Developer (alphabetical)

- Liam Huber (MPIE) - Leading the implementation of flexible simulation protocols - since 2019.

- Jan Janssen (MPIE) – Leading the pyiron development – since 2015.
- Sudarsan Surendralal (MPIE) – Leading the electronic structure code development – since 2015.
- Osamu Waseda (MPIE) – Leading the run-time coupling of simulation codes – since 2017.

4.4.2 Application Developer (alphabetical)

- Ahmed Aslam (MPIE) - Parameterisation of interatomic potentials - since 2018.
- Uday Gajera (MPIE) - Automated analysis of existing DFT data sets - since 2017.
- Yury Lysogorski (ICAMS) – High throughput evaluation of interatomic potentials – since 2017.
- Lifang Zhu (MPIE) - Efficient approach to compute melting properties fully from ab initio - since 2017.

4.4.3 Steering Committee (Head: Joerg Neugebauer)

- Joerg Neugebauer (MPIE) – Founding and lead developer - since 2010
- Mira Todorova (MPIE), Christoph Freysoldt (MPIE) – Electronic structure features
- Tilmann Hickel (MPIE), Blazej Grabowski (MPIE) – Thermodynamic projects - Thermodynamic concepts
- Ralf Drautz (ICAMS), Thomas Hammerschmidt (ICAMS) – High-throughput activities

4.4.4 Alumni (chronological)

- Ugur Aydin (MPIE) – Developer of pyCMW – the pyiron predecessor – 2011-2015.
- Ankit Gupta (MPIE) – Kinetic Monte Carlo implementation – 2014-2015.
- Murat Celik (MPIE) – Sqlalchemy based database adapter – 2016-2017.
- Navid Shayanfar (MPIE) - Parser for the in-house DFT code S/PHI/nX - 2017.
- Martin Boeckmann (MPIE) – Metropolis Monte Carlo implementation – 2017-2018.
- Murali Uddagiri (MPIE) - Generation of special quasirandom structures - 2017-2018.
- Markus Tautschnig (MPIE) – Structure Optimisation with VASP – 2018-2019.

4.4.5 External collaborators

- Max Planck Computing & Data facility (MPCDF) - The MPCDF provides high-level support for the development, optimization, analysis and visualization of high-performance-computing applications.

4.5 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.

4.5.1 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.

4.5.2 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},
```

(continues on next page)

(continued from previous page)

```
  month = {Oct},
  publisher = {American Physical Society},
  doi = {10.1103/PhysRevB.54.11169},
  url = {https://link.aps.org/doi/10.1103/PhysRevB.54.11169}
}
```

4.6 FAQ

4.6.1 How to cite pyiron?

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

4.6.2 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)

4.6.3 How to import existing calculation?

4.6.4 How to import structures from files or existing databases?

4.6.5 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.

4.6.6 How to use a custom Pseudo potential in VASP?

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

4.6.8 How to use a custom potential in LAMMPS?

4.6.9 How to extend the potential database inside pyiron?

4.6.10 How to link your own executable?

4.6.11 How to send a calculation to the background ?

4.6.12 How to submit a calculation to the queuing system?

4.6.13 How to setup spin constraint calculation?

4.6.14 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).

4.6.15 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