The LatticeFinder Documentation

Release 1

Geoffrey Weal

Aug 11, 2021

CONTENTS

[1] [2] [3] [4] [5] [6] [7] *Section author: Geoffrey Weal <geoffrey.weal@gmail.com>*

Section author: Dr. Anna Garden <anna.garden@otago.ac.nz>

Group page: https://blogs.otago.ac.nz/annagarden/

[1] https://docs.python.org/3/
[2] https://github.com/GardenGroupUO/LatticeFinder
[3] https://pypi.org/project/LatticeFinder/
[4] https://anaconda.org/GardenGroupUO/latticefinder
[5] https://mybinder.org/v2/gh/GardenGroupUO/LatticeFinder/main?urlpath=lab
[6] https://www.gnu.org/licenses/agpl-3.0.en.html
[7] https://lgtm.com/projects/g/GardenGroupUO/LatticeFinder/context:python

CHAPTER

ONE

WHAT IS LATTICEFINDER

LatticeFinder is a program for finding the optimum lattice constant for a bulk 3D system (and at some point 2D systems as well).

CHAPTER

TWO

TRY LATTICEFINDER BEFORE YOU CLONE/PIP/CONDA (ON BINDER/JUPTER NOTEBOOKS)!

If you are new to the LatticeFinder program, it is recommended try it out by running LatticeFinder live on our interactive Jupyter+Binder page before you download it. On Jupyter+Binder, you can play around with the LatticeFinder program on the web. You do not need to install anything to try LatticeFinder out on Jupyter+Binder.

Click the Binder button below to try LatticeFinder out on the web! (The Binder page may load quickly or may take 1 or 2 minutes to load)

[8]

[8] https://mybinder.org/v2/gh/GardenGroupUO/Organisms_Jupyter_Examples/main?urlpath=lab

CHAPTER

THREE

INSTALLATION

It is recommended to read the installation page before using the LatticeFinder program. See *Installation: Setting Up the LatticeFinder Program and Pre-Requisites Packages* for more information. Note that you can install LatticeFinder through `pip3` and `conda`.

CHAPTER

FOUR

TABLE OF CONTENTS

4.1 How the LatticeFinder program works

This program is designed to help you obtain the optimal lattice constants for a 2D system (such as a graphene based system) or a 3D based system (such as a face-centered cubic lattice), as well as other parameters of the system with optimal lattice constants, such as the bulk modulus.

4.2 Installation: Setting Up LatticeFinder and Pre-Requisites Packages

In this article, we will look at how to install the LatticeFinder and all requisites required for this program.

4.2.1 Pre-requisites

Python 3 and `pip3`

This program is designed to work with **Python 3**. While this program has been designed to work with Python 3.6, it should work with any version of Python 3 that is the same or later than 3.6.

To find out if you have Python 3 on your computer and what version you have, type into the terminal

```
python3 --version
```

If you have Python 3 on your computer, you will get the version of python you have on your computer. E.g.

```
geoffreyweal@Geoffreys-Mini Documentation % python3 --version
Python 3.6.3
```

If you have Python 3, you may have `pip3` installed on your computer as well. `pip3` is a python package installation tool that is recommended by Python for installing Python packages. To see if you have `pip3` installed, type into the terminal

```
pip3 list
```

If you get back a list of python packages install on your computer, you have `pip3` installed. E.g.

```
geoffreyweal@Geoffreys-Mini Documentation % pip3 list
Package                       Version
----------------------------- ---------
```

(continues on next page)

(continued from previous page)

```
alabaster                       0.7.12
asap3                           3.11.10
ase                             3.20.1
Babel                           2.8.0
certifi                         2020.6.20
chardet                         3.0.4
click                           7.1.2
cycler                          0.10.0
docutils                        0.16
Flask                           1.1.2
idna                            2.10
imagesize                       1.2.0
itsdangerous                    1.1.0
Jinja2                          2.11.2
kiwisolver                      1.2.0
MarkupSafe                      1.1.1
matplotlib                      3.3.1
numpy                           1.19.1
packaging                       20.4
Pillow                          7.2.0
pip                             20.2.4
Pygments                        2.7.1
pyparsing                       2.4.7
python-dateutil                 2.8.1
pytz                            2020.1
requests                        2.24.0
scipy                           1.5.2
setuptools                      41.2.0
six                             1.15.0
snowballstemmer                 2.0.0
Sphinx                          3.2.1
sphinx-pyreverse                0.0.13
sphinx-rtd-theme                0.5.0
sphinx-tabs                     1.3.0
sphinxcontrib-applehelp         1.0.2
sphinxcontrib-devhelp           1.0.2
sphinxcontrib-htmlhelp          1.0.3
sphinxcontrib-jsmath            1.0.1
sphinxcontrib-plantuml          0.18.1
sphinxcontrib-qthelp            1.0.3
sphinxcontrib-serializinghtml 1.1.4
sphinxcontrib-websupport        1.2.4
urllib3                         1.25.10
Werkzeug                        1.0.1
wheel                           0.33.1
xlrd                            1.2.0
```

If you do not see this, you probably do not have `pip3` installed on your computer. If this is the case, check out PIP Installation[9]

[9] https://pip.pypa.io/en/stable/installing/

Atomic Simulation Environment

LatticeFinder uses the atomic simulation environment (ASE) to create models of crystal structures. This allows NISP to take advantage of the features of ASE, such as the wide range of calculators that can be used to calculate the energy of the cluster. Furthermore, ASE also offers useful tools for viewing, manipulating, reading and saving clusters and chemcial systems easily. Read more about ASE here[10]. For NISP, it is recommended that you **install a version of ase that is 3.19.1 or greater**.

The installation of ASE can be found on the ASE installation page[11], however from experience if you are using ASE for the first time, it is best to install ASE using pip, the package manager that is an extension of python to keep all your program easily managed and easy to import into your python.

To install ASE using pip, perform the following in your terminal.

```
pip3 install --upgrade --user ase
```

Installing using `pip3` ensures that ASE is being installed to be used by Python 3, and not Python 2. Installing ASE like this will also install all the requisite program needed for ASE. This installation includes the use of features such as viewing the xyz files of structure and looking at ase databases through a website. These should be already assessible, which you can test by entering into the terminal:

```
ase gui
```

This should show a gui with nothing in it, as shown below.

However, in the case that this does not work, we need to manually add a path to your `~/.bashrc` so you can use the ASE features externally outside python. First enter the following into the terminal:

```
pip3 show ase
```

This will give a bunch of information, including the location of ase on your computer. For example, when I do this I get:

```
Geoffreys-Mini:~ geoffreyweal$ pip show ase
Name: ase
Version: 3.20.1
Summary: Atomic Simulation Environment
Home-page: https://wiki.fysik.dtu.dk/ase
Author: None
Author-email: None
License: LGPLv2.1+
Location: /Users/geoffreyweal/Library/Python/3.6/lib/python/site-packages
Requires: matplotlib, scipy, numpy
Required-by:
```

In the 'Location' line, if you remove the 'lib/python/site-packages' bit and replace it with 'bin'. The example below is for Python 3.6.

```
/Users/geoffreyweal/Library/Python/3.6/bin
```

This is the location of these useful ASE tools. You want to put this as a path in your `~/.bashrc` as below:

```
############################################################
# For ASE
export PATH=/Users/geoffreyweal/Library/Python/3.6/bin:$PATH
############################################################
```

[10] https://wiki.fysik.dtu.dk/ase/
[11] https://wiki.fysik.dtu.dk/ase/install.html

Fig. 1: This is a blank ase gui screen that you would see if enter `ase gui` into the terminal.

Packaging

The packaging program is also used in this program to check the versions of ASE that you are using for compatibility issues. Easiest way to install packaging is though pip. Type the following into the terminal:

```
pip3 install --upgrade --user packaging
```

4.2.2 Setting up LatticeFinder

There are three ways to install LatticeFinder on your system. These ways are described below:

Install LatticeFinder through `pip3`

To install the LatticeFinder program using `pip3`, perform the following in your terminal.

```
pip3 install --upgrade --user LatticeFinder
```

The website for LatticeFinder on `pip3` can be found by clicking the button below:

[12]

Install LatticeFinder through `conda`

You can also install LatticeFinder through `conda`, however I am not as versed on this as using `pip3`. See docs.conda.io[13] to see more information about this. Once you have installed anaconda on your computer, I believe you install LatticeFinder using `conda` by performing the following in your terminal.

```
conda install ase
conda install latticefinder
```

The website for LatticeFinder on `conda` can be found by clicking the button below:

[14]

Manual installation

First, download LatticeFinder to your computer. You can do this by cloning a version of this from Github, or obtaining a version of the program from the authors. If you are obtaining this program via Github, you want to `cd` to the directory that you want to place this program in on the terminal, and then clone the program from Github through the terminal as well

```
cd PATH/TO/WHERE_YOU_WANT_LatticeFinder_TO_LIVE_ON_YOUR_COMPUTER
git clone https://github.com/GardenGroupUO/LatticeFinder
```

Next, add a python path to it in your `.bashrc` to indicate its location. Do this by entering into the terminal where you cloned the LatticeFinder program into `pwd`

```
pwd
```

[12] https://pypi.org/project/LatticeFinder/
[13] https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html
[14] https://anaconda.org/GardenGroupUO/latticefinder

This will give you the path to the LatticeFinder program. You want to enter the result from `pwd` into the `.bashrc` file. This is done as shown below:

```
export PATH_TO_LatticeFinder="<Path_to_LatticeFinder>"
export PYTHONPATH="$PATH_TO_LatticeFinder":$PYTHONPATH
```

where `"<Path_to_LatticeFinder>"` is the directory path that you place LatticeFinder (Enter in here the result you got from the `pwd` command). Once you have run `source ~/.bashrc`, the genetic algorithm should be all ready to go!

The folder called `Examples` contains all the files that one would want to used to use the genetic algorithm for various metals. This includes examples of the basic run code for the genetic algorithm, the `Interpolation_Script.py` and `RunMinimisation.py` files.

LatticeFinder contains subsidiary programs that contain other program that may be useful to use when using the LatticeFinder program. This is called `Subsidiary_Programs` in LatticeFinder. To execute any of the programs contained within the `Subsidiary_Programs` folder, include the following in your `~/.bashrc`:

```
export PATH="$PATH_TO_LatticeFinder"/LatticeFinder/Subsidiary_Programs:$PATH
```

Other Useful things to know before you start

You may use squeue to figure out what jobs are running in slurm. For monitoring what slurm jobs are running, I have found the following alias useful. Include the following in your `~/.bashrc`

```
squeue -o "%.20i %.9P %.5Q %.50j %.8u %.8T %.10M %.11l %.6D %.4C %.6b %.20S %.20R %.8q
↪" -u $USER --sort=+i
```

Summary of what you want in the `~/.bashrc` for the LatticeFinder program if you manually installed LatticeFinder

You want to have the following in your `~/.bashrc`:

```
#########################################################
# Paths and Pythonpaths for LatticeFinder

export PATH_TO_LatticeFinder="<Path_to_LatticeFinder>"
export PYTHONPATH="$PATH_TO_LatticeFinder":$PYTHONPATH

export PATH="$PATH_TO_LatticeFinder"/LatticeFinder/Subsidiary_Programs:$PATH

squeue -o "%.20i %.9P %.5Q %.50j %.8u %.8T %.10M %.11l %.6D %.4C %.6b %.20S %.20R %.8q
↪" -u $USER --sort=+i

#########################################################
```

4.3 *Run_LatticeFinder.py* - How to run LatticeFinder

In this article, we will look at how to run LatticeFinder. LatticeFinder is run through the `Run_LatticeFinder.py` python script. You can find examples of `Run_LatticeFinder.py` files at github.com/GardenGroupUO/LatticeFinder[15] under `Examples`. Also, you can try out this program by running an example script through a Jupyter notebook. See *Examples of running LatticeFinder* to get access to examples of running LatticeFinder through this Jupyter notebook!

4.3.1 Running the `Run_LatticeFinder.py` script

We will explain how the `Run_LatticeFinder.py` code works by running though the example shown below:

Listing 1: Run_LatticeFinder.py

```
from LatticeFinder import LatticeFinder_Program
import numpy as np

symbol = 'Au'
lattice_type = 'FaceCenteredCubic'

lattice_constant_parameters = list(np.arange(3.0,3.8,0.1))+list(np.arange(3.8,4.5,0.
↪01))+list(np.arange(4.5,5.01,0.1))

from asap3.Internal.BuiltinPotentials import Gupta
# Parameter sequence: [p, q, a, xi, r0]
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
cutoff = 8
calculator = Gupta(Au_parameters, cutoff=cutoff, debug=False)

size = (16,16,16)

directions = []
miller = []

limit = None
make_svg_eps_files = False

no_of_cpus = 1

LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, limit=limit, make_svg_eps_
↪files=make_svg_eps_files, no_of_cpus=no_of_cpus, slurm_information=slurm_
↪information)
```

Lets go through each part of the `Run_LatticeFinder.py` file one by one to understand how to use it.

[15] https://github.com/GardenGroupUO/LatticeFinder

Input information for LatticeFinder

The following information is required by LatticeFinder:

- **symbol** (*str.*): This is the element that makes up your 2D/3D system.
- **lattice_type** (*str.*): This is the type of lattice that you which to obtain the optimal lattice constants for. See Available crystal lattices in ASE[16] for more information.
- **lattice_constant_parameters** (*list of floats*): These are the values of the lattice constant(s) that you would like to examine. There are two ways that this can be entered into LatticeFinder:
  - If you are locating the optimal lattice constant for a system that only contains one lattice constant, this can be entered in as a `np.arange`. For example, if you want to scan between a lattice constant of 3.0 Å to 5.0 Å in 0.1 Å segments, `lattice_constant_parameters = np.arange(3.0,5.01,0.1)`. You can also have an irregular list. For example, if you want to look further between 3.8 Å and 4.5 Å in 0.01 segments, you can set `lattice_constant_parameters = list(np.arange(3.0,3.8,0.1))+list(np.arange(3.8,4.5,0.01))+list(np.arange(4.5,5.01,0.1))`
  - If you are locating the optimal lattice constanta for a system that only contains two lattice constants, this must be entered as a list, where each key is the name of the lattice constant. For example, for a hexagonal closed packed crystal, you can set `lattice_constant_parameters = {'a': np.arange(2.0,5.01,0.1), 'c': np.arange(3.0,6.01,0.1)}`. The lists for each lattice constant must be regularly spaced; you can not use irregular spacing in LatticeFinder for system with more than one lattice constant.
- **calculator** (*ase.calculator/str.*): The calculator is used to calculate the energy of the 2D/3D system at various lattice constants. See Calculators in ASE[17] for information about how calculators works in ASE.
  - You can also use **VASP to perform DFT local optimisations** on your clusters. Do this by setting `calculator = 'VASP'`. See *How to perform LatticeFinder with VASP calculations* to learn more about how to perform VASP calculations on clusters created using NISP.
  - You can also elect to **manually enter in the energies of the clusters**. To do this, enter in `calculator = 'Manual Mode'`. See *How to manually enter energy results into LatticeFinder* for more information about how to manually enter in energies for clusters into LatticeFinder.
- **size** (*list of ints*): This is the size of the system within a cell. See Usage in Lattices[18] for more infotmation about the size parameter.
- **directions** (*list of ints*): Still figuring this out. See Usage in Lattices[19] for more infotmation about the directions parameter.
- **miller** (*list of ints*): Still figuring this out. See Usage in Lattices[20] for more infotmation about the miller parameter.

Parameters required by LatticeFinder for plotting plots:

- **limit** (*dict.*): This is the limits for plotting your lattice constant plots. For a lattice system with one lattice constant: give as `{'c': (c_lower_limit, c_upper_limit)}`. For a lattice system with two lattice constant: give as `{'c': (c_lower_limit, c_upper_limit), 'a': (a_lower_limit, a_upper_limit)}`. If no change to the plotting limits are needed, set this to `None`. Default: `None`.
- **make_svg_eps_files** (*bool*): This tag tells LatticeFinder if you want to create svg and eps files of the plots made. Default: `True`.

Other parameters required by LatticeFinder:

[16] https://wiki.fysik.dtu.dk/ase/ase/lattice.html#available-crystal-lattices
[17] https://wiki.fysik.dtu.dk/ase/ase/calculators/calculators.html
[18] https://wiki.fysik.dtu.dk/ase/ase/lattice.html#usage
[19] https://wiki.fysik.dtu.dk/ase/ase/lattice.html#usage
[20] https://wiki.fysik.dtu.dk/ase/ase/lattice.html#usage

- **no_of_cpus** (*int*): This is the number of cpus that you would like to use to perform calculations of 2D/3D system of various lattice constants.

An example of these parameters in `Run_LatticeFinder.py` is given below:

```
from LatticeFinder import LatticeFinder_Program
import numpy as np

symbol = 'Au'
lattice_type = 'FaceCenteredCubic'

lattice_constant_parameters = list(np.arange(3.0,3.8,0.1))+list(np.arange(3.8,4.5,0.
↪01))+list(np.arange(4.5,5.01,0.1))

from asap3.Internal.BuiltinPotentials import Gupta
# Parameter sequence: [p, q, a, xi, r0]
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
cutoff = 8
calculator = Gupta(Au_parameters, cutoff=cutoff, debug=False)

size = (16,16,16)

directions = []
miller = []

limit = None
make_svg_eps_files = False

no_of_cpus = 1
```

Run LatticeFinder!

You have got to the end of all the parameter setting stuff. Now on to running NISP. The next part of the `Run_LatticeFinder.py` script tells NISP to run. This is written as follows in the `Run_LatticeFinder.py`:

```
LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, limit=limit, make_svg_eps_
↪files=make_svg_eps_files, no_of_cpus=no_of_cpus, slurm_information=slurm_
↪information)
```

4.3.2 Output files that are created by LatticeFinder

The LatticeFinder program will create a number of plots and text documents when it is run. See *Examples of Running LatticeFinder with Run_LatticeFinder.py* and LatticeFinder examples here[21] to see the types of plots and text documents that LatticeFinder will make.

[21] https://github.com/GardenGroupUO/LatticeFinder/tree/main/Examples

4.4 How to perform LatticeFinder with VASP calculations

In this article, we will look at how to run LatticeFinder where VASP is used to calculate the energies of systems at various lattice constants. The `Run_LatticeFinder.py` python script that is used is the same as shown previously in *Run_LatticeFinder.py - How to run LatticeFinder*, but with an extra component. An example of a `Run_LatticeFinder.py` python script that uses VASP is shown below:

Listing 2: Run_LatticeFinder.py

```
from LatticeFinder import LatticeFinder_Program
import numpy as np

symbol = 'Au'
lattice_type = 'FaceCenteredCubic'

lattice_constant_parameters = (2.0,6.0,0.1)

calculator = 'VASP'
slurm_information = {}
slurm_information['project'] = 'uoo02568'
slurm_information['time'] = '2:00:00'
slurm_information['nodes'] = 1
slurm_information['ntasks_per_node'] = 8
slurm_information['mem-per-cpu'] = '3G'
slurm_information['partition'] = 'large'
slurm_information['email'] = 'geoffreywealslurmnotifications@gmail.com'
slurm_information['python_version'] = 'Python/3.6.3-gimkl-2017a'
slurm_information['vasp_version'] = 'VASP/5.4.4-intel-2017a'
slurm_information['vasp_execution'] = 'vasp_std'

slurm_information['Make individual or packet submitSL files'] = 'packets'
slurm_information['Number of VASP calculations to run per packet'] = 25

size=(1,1,1)

directions = []
miller = []

limit = None
make_svg_eps_files = False

no_of_cpus = 1

LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, limit=limit, make_svg_eps_
↪files=make_svg_eps_files, no_of_cpus=no_of_cpus, slurm_information=slurm_
↪information)
```

4.4.1 The `slurm_information` parameter

The extra parameter that is included when performing VASP calculations is the `slurm_information` parameter, which is a dictionary that holds all the information that is needed to create the `submit.sl` files required to submit VASP calculations to slurm. The following information is needed in the `'slurm_information'` dictionary:

- **project** (*str.*): This is the name of the project that you want to submit this job to.
- **time** (*str.*): This is the amount of time you want to give to your slurm jobs, given as `'HH:MM:SS'`, where `'HH:MM:SS'` is the hours, minutes, and seconds you want to give to a job.
- **nodes** (*str.*): This is the number of nodes that you would like to give to a job.
- **ntasks_per_node** (*str./int*): This is the number of cpus that you give to a job.
- **mem-per-cpu** (*str.*): This is the amount of momeory you are giving to your job per cpu

The following can also be included in `'slurm_information'` dictionary, but these are default value for these if you do not give a value for them.

- **partition** (*str.*): This is the partition that is given to your job. See Mahuika Slurm Partitions[22] for more information about partition on NeSI (Default: `'large'`).
- **email** (*str.*): This is the email address you would like notifications about your slurm job to be sent to (Default: `''`).
- **vasp_version** (*str.*): This is the version of VASP that you would like to load in on slurm (Default: `'VASP/5.4.4-intel-2017a'`).
- **vasp_execution** (*str.*): This is the name of the vasp program that you execute (Default: `'vasp_std'`).

Commonly in VASP you will only need to use a (1,1,1) cell since VASP performs calculations with periodic boundary conditions. Because of this, you will be performing many small VASP calculations. As you may be performing many short VASP calculations, it is possible to break the slurm management system as slurm can get confused when its accepts many jobs at once which then finish very quickly. NeSI (support.nesi.org.nz) suggest that you should instead run packets of short VASP calculation in serial to minimise this happening. In this case, there are two additional setting to give the `slurm_information` dictionary:

- **Make individual or packet submitSL files** (*str.*): Determines how jobs are submitted to slurm. If `'individual'`: a slurm.sl file is created for each VASP job to run; if `'packet'`: Several individual VASP jobs will be packaged together and run one after the other (serial) in slurm (Default: `'individual'`).
- **Number of VASP calculations to run per packet** (*int*): If you choose `slurm_information['Make individual or packet submitSL files'] = 'packets'`, this is the number of individual VASP jobs that will be packaged together and run one after the other (serial) in slurm.

See an example of the `slurm_information` parameter below:

Listing 3: Run_LatticeFinder.py

```
10 slurm_information = {}
11 slurm_information['project'] = 'uoo02568'
12 slurm_information['time'] = '2:00:00'
13 slurm_information['nodes'] = 1
14 slurm_information['ntasks_per_node'] = 8
15 slurm_information['mem-per-cpu'] = '3G'
16 slurm_information['partition'] = 'large'
17 slurm_information['email'] = 'geoffreywealslurmnotifications@gmail.com'
18 slurm_information['python_version'] = 'Python/3.6.3-gimkl-2017a'
19 slurm_information['vasp_version'] = 'VASP/5.4.4-intel-2017a'
```

(continues on next page)

[22] https://support.nesi.org.nz/hc/en-gb/articles/360000204076-Mahuika-Slurm-Partitions

(continued from previous page)

```
slurm_information['vasp_execution'] = 'vasp_std'

slurm_information['Make individual or packet submitSL files'] = 'packets'
slurm_information['Number of VASP calculations to run per packet'] = 25
```

Make sure that you include `'slurm_information'` in the final line of `Run_LatticeFinder.py` in `LatticeFinder_Program`. See the following code before to see this:

Listing 4: Run_LatticeFinder.py

```
LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, limit=limit, make_svg_eps_
↪files=make_svg_eps_files, no_of_cpus=no_of_cpus, slurm_information=slurm_
↪information)
```

4.4.2 Other files that you will need

You will also need to give LatticeFinder some other files that are needed by VASP to perform calculations. In the same place where you place your `Run_LatticeFinder.py` file, you want to create another folder called `VASP_Files`. In this `VASP_Files` folder you want to include the following files:

- `POTCAR`: This is the file that contains the information required to locally optimise a nanocluster with DFT using a certain functional.
- `KPOINTS`: This contain the information used to specify the Bloch vectors (k-points) that will be used to sample the Brillouin zone in your calculation.
- `INCAR`: This contains all the setting that are required by VASP to perform calculations. **Note that in the INCAR you must set** `NSW = 0`. This prevents VASP from performing a local optimisation which you do not need to do in this program.

These files will be copied by LatticeFinder into each nanocluster folder. See an example of a setup of LatticeFinder for VASP here[23].

4.4.3 What to do after you have run LatticeFinder

After you run LatticeFinder, this will create a new folder called `VASP_Systems`, which contains subfolders of your system at all the lattice constants that you want to examine. Each subfolder will contain a `POSCAR`, `INCAR`, `POTCAR`, `KPOINTS`, and `submit.sl` that are needed by VASP to perform DFT calculations. Each system is ready to be calculated by VASP.

You will find that there are many systems are created by LatticeFinder. To submit all of these system to slurm to calculate energies for by VASP, you can execute the program called `Run_LatticeFinder_submitSL_slurm.py` which will execute all of DFT VASP jobs in slurm. To run this script, type `Run_submitSL_slurm.py` into the terminal inside of your newly created `VASP_Systems` folder.

[23] https://github.com/GardenGroupUO/LatticeFinder/tree/main/Examples/Uncompleted_Examples/VASP_Au_FCC_Full

Output files that are created by LatticeFinder

The LatticeFinder program will create a number of plots and text documents when it is run. See *Examples of Running LatticeFinder with Run_LatticeFinder.py* and LatticeFinder examples here[24] to see the types of plots and text documents that LatticeFinder will make.

4.5 How to manually enter energy results into LatticeFinder

If this is desired, add a message on github to develop this part of the code.

4.6 A guide for efficiently using LatticeFinder to obtain the optimal lattice constants

In this article, we will describe how to use LatticeFinder to find the optimal lattice constants for a 2D or 3D system. We will describe how to efficiently use LatticeFinder for systems that contain either one lattice constant or two lattice constants.

4.6.1 2D and 3D Systems containing one lattice constant

1) Performing a broad overview of lattice constants

To begin it is often useful to perform a broad overview of lattice constants to determine what lattice constants to focus on. For example, for determining the lattice constant of a Au face-centred cubic (FCC) lattice using the RGL potential, with parameters from Baletto *et al.* (DOI: 10.1063/1.1448484[25]). Here, we first look at lattice constants between 3.0 Å and 5.0 Å in increments of 0.1 Å. This is performed with the following `Run_LatticeFinder.py` script:

Listing 5: Run_LatticeFinder.py

```
from LatticeFinder import LatticeFinder_Program

symbol = 'Au'
lattice_type = 'FaceCenteredCubic'

lattice_constant_parameters = (3.0,5.0,0.1)

from asap3.Internal.BuiltinPotentials import Gupta
# Parameter sequence: [p, q, a, xi, r0]
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
cutoff = 8
calculator = Gupta(Au_parameters, cutoff=cutoff, debug=False)

size=(16,16,16)

directions=[]
miller=[]

limits = None
```

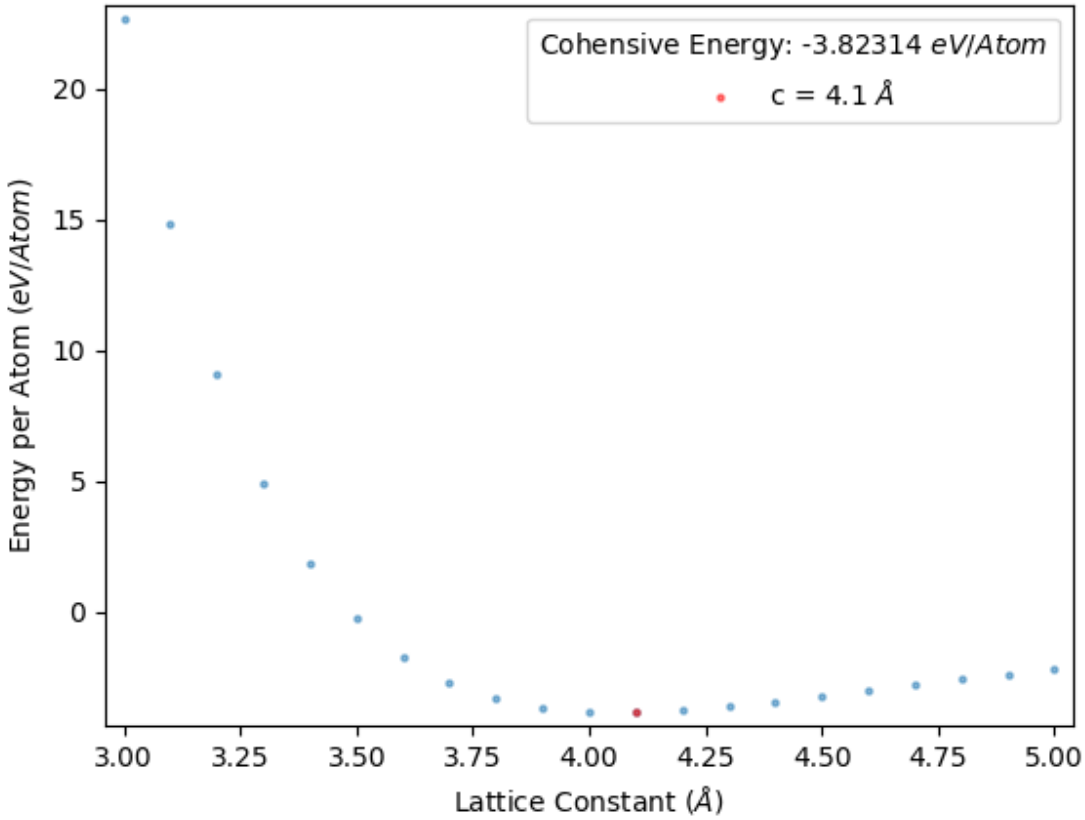
(continues on next page)

[24] https://github.com/GardenGroupUO/LatticeFinder/tree/main/Examples
[25] https://doi.org/10.1063/1.1448484

(continued from previous page)

```
21
22 no_of_cpus = 2
23
24 LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
   ↪size=size, directions=directions, miller=miller, limits=limits, no_of_cpus=no_of_
   ↪cpus)
```

This gives an energy per atom vs lattice constant plot as below:



From this plot, we see that the minimum centres about 4.1 Å. Therefore, we will want to next perform a comprehensive search of the optimum lattice constant between 4.0 Å and 4.2 Å at the least. This is because we set this script to scan the lattic constants in increments of 0.1 Å, therefore the precise lattice constant will lie within the 4.1 ± 0.1 Å range.

2) Performing a comprehensive scan of lattice constants across a small range

We would like to perform a indepth scan of lattice constants between 4.0 Å and 4.2 Å. We will do this by looking between 4.0 Å and 4.2 Å in increments of 0.001 Å. We can add this indepth scan to our original LatticeFinder calculation by changing the `Run_LatticeFinder.py` script to the following:

Listing 6: Run_LatticeFinder.py

```
from LatticeFinder import LatticeFinder_Program
import numpy as np

symbol = 'Au'
lattice_type = 'FaceCenteredCubic'

lattice_constant_parameters = list(np.arange(3.0,4.0,0.1))+list(np.arange(4.0,4.2,0.
↪001))+list(np.arange(4.2,5.01,0.1))

from asap3.Internal.BuiltinPotentials import Gupta
# Parameter sequence: [p, q, a, xi, r0]
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
cutoff = 8
calculator = Gupta(Au_parameters, cutoff=cutoff, debug=False)

size=(16,16,16)

directions=[]
miller=[]
no_of_cpus=1

LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, no_of_cpus=no_of_cpus)
```

You can change the `lattice_constant_parameters` input from

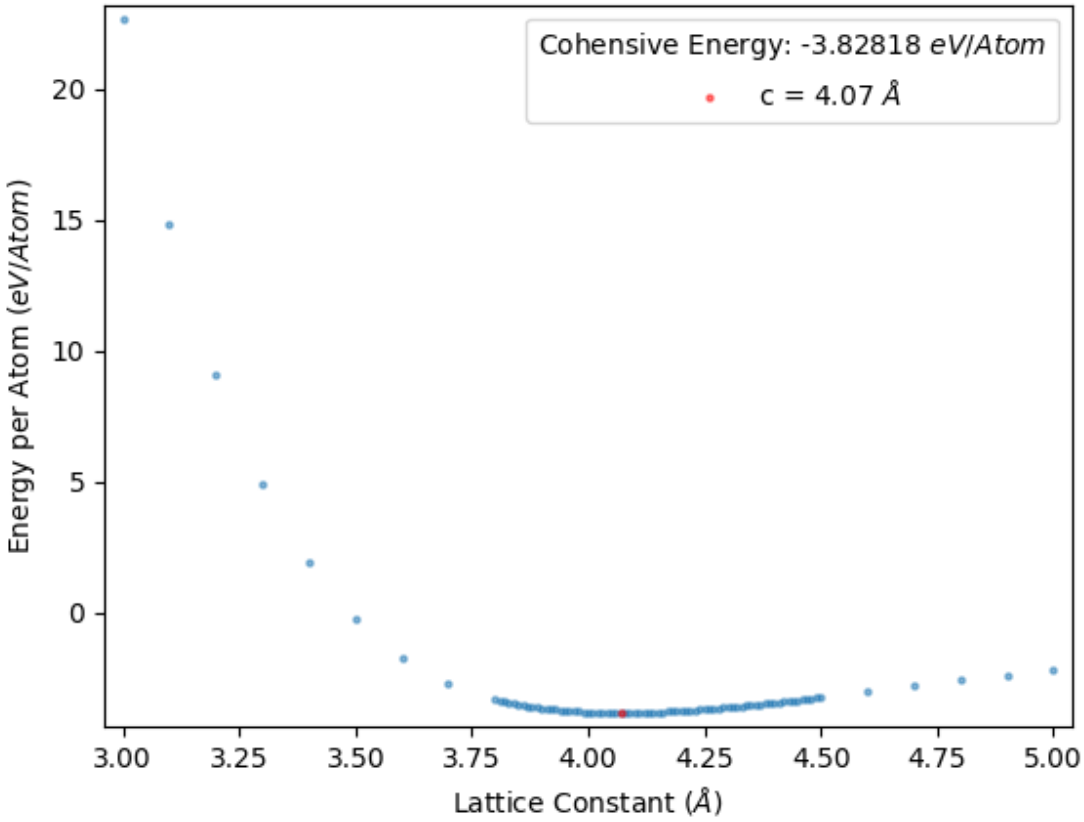
```
lattice_constant_parameters = (3.0,5.0,0.1)
```

to

```
lattice_constant_parameters = list(np.arange(3.0,4.0,0.1))+list(np.arange(4.0,4.2,0.
↪001))+list(np.arange(4.2,5.01,0.1))
```

and rerun LatticeFinder. LatticeFinder will perform calculations with lattice constants that you have not already obtained. Once you rerun LatticeFinder on your script again, you will get the following energy per atom vs lattice constant plot as below:

This gives an energy per atom vs lattice constant plot as below:



The data from this is shown in `results_file.txt`

```
Symbol: Au
Lattice_type: FaceCenteredCubic
calculator: <asap.RGL object at 0x0x54b110>
size: (16, 16, 16)
directions: []
miller: []
Lattice Constant Parameters: ['c']

Properties of System:

Total energy: -62720.92122649953 eV
No. of atoms: 16384 Atoms (Note the number of atoms along each natural direction of␣
↪the bulk is (16, 16, 16))
Cohesive energy: -3.8281812272033404 eV/Atom

Total Volume: 276148.8097280001 Angstroms^3
Volume per atom: 16.854785750000005 Angstroms^3/Atom

Stress tensor:
[[1.29870561e-03 1.29377728e-19 1.68652768e-19]
 [1.29377728e-19 1.29870561e-03 1.29962170e-19]
 [1.68652768e-19 1.29962170e-19 1.29870561e-03]]

Bulk Modulus: 218.1717839967987 GPa
```

How to obtain the Bulk Modulus

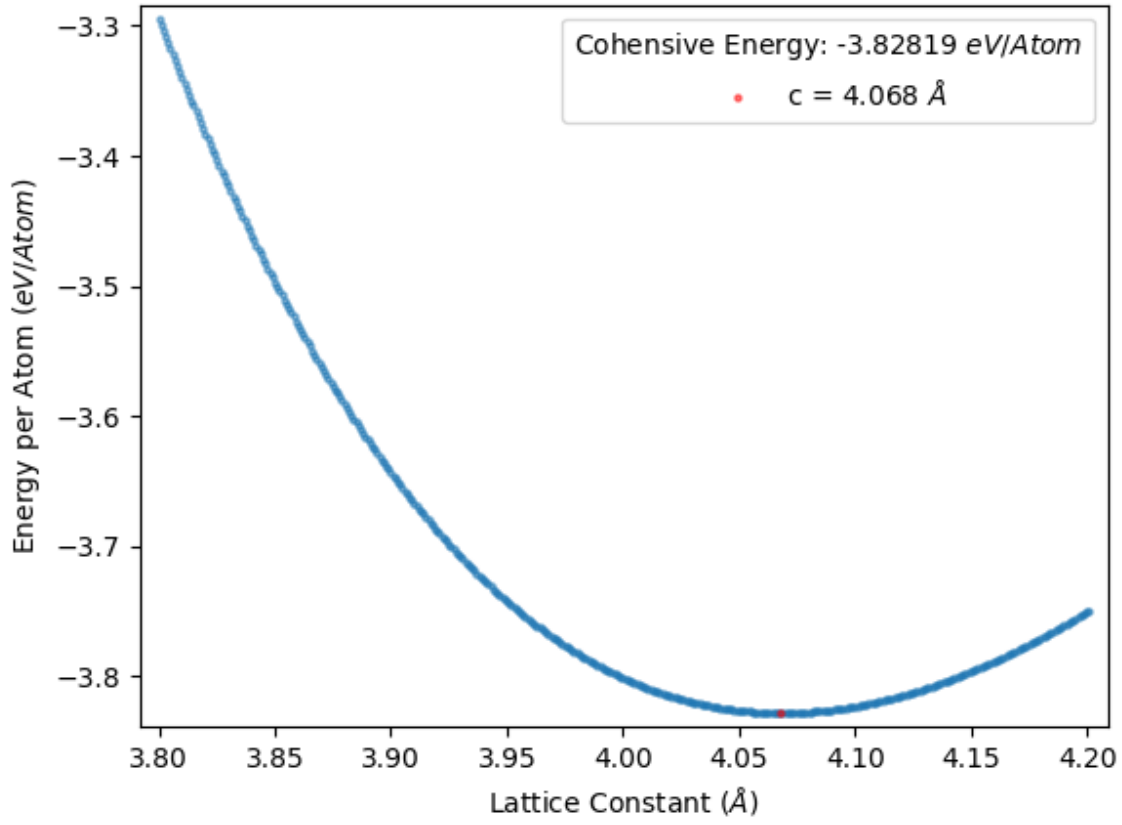
In your `results_file.txt` you will also be given a value for the bulk modulus. If you want the bulk modulus with a good amount of accurancy, you often need to perform an indepth scan of lattice constants across a good range of data points. Furthermore, the range of lattice constants given must be regular, i.e. incremental.

For example, we have obtain the energies of an Au FCC crystal for lattice constants between 3.6 Å and 4.6 Å in increments of 0.001 Å. This is given in the `Run_LatticeFinder.py` script below:

Listing 7: Run_LatticeFinder.py

```
from LatticeFinder import LatticeFinder_Program

symbol = 'Au'
lattice_type = 'FaceCenteredCubic'

lattice_constant_parameters = (3.6,4.6,0.001)

from asap3.Internal.BuiltinPotentials import Gupta
# Parameter sequence: [p, q, a, xi, r0]
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
cutoff = 8
calculator = Gupta(Au_parameters, cutoff=cutoff, debug=False)

size=(16,16,16)

directions=[]
miller=[]
no_of_cpus=1

LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, no_of_cpus=no_of_cpus)
```

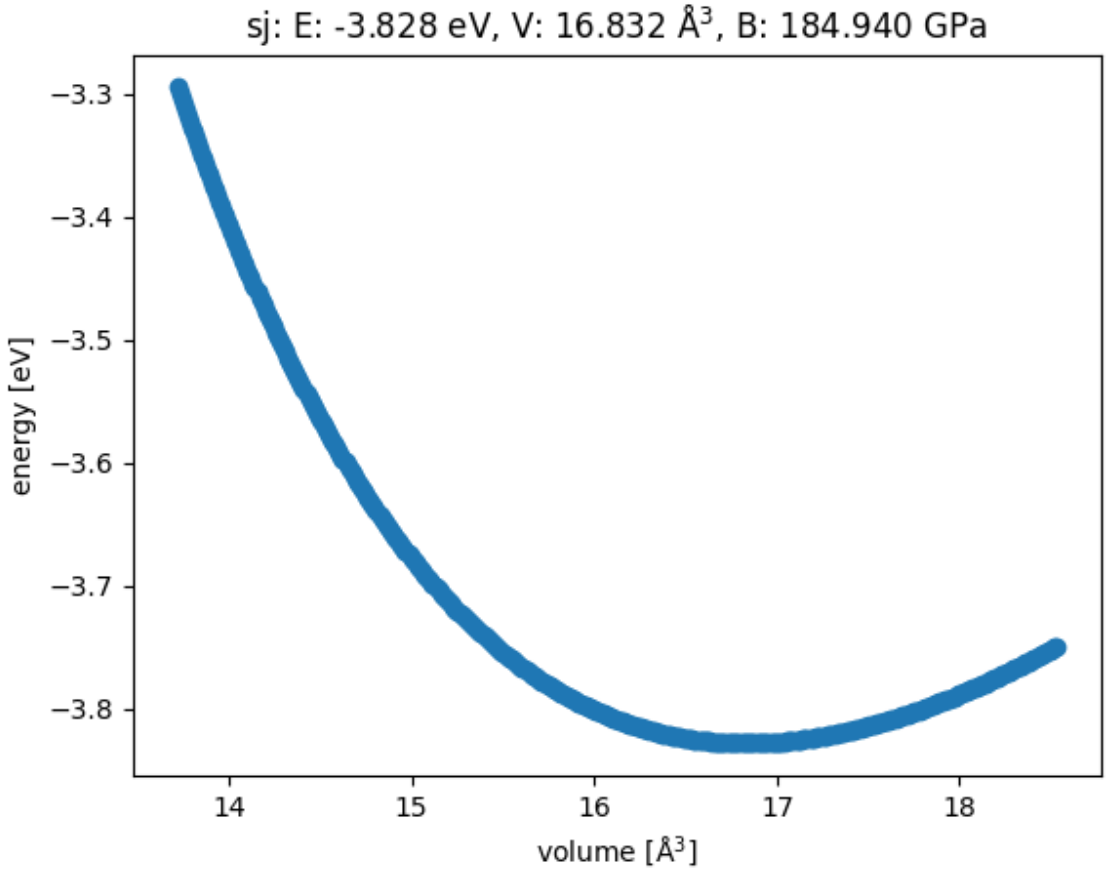
This gives the following energy per atom vs lattice constant plot:



and the data is given in `results_file.txt`

```
Symbol: Au
Lattice_type: FaceCenteredCubic
calculator: <asap.RGL object at 0x0x54b110>
size: (16, 16, 16)
directions: []
miller: []
Lattice Constant Parameters: ['c']

Properties of System:

Total energy: -62720.92122649953 eV
No. of atoms: 16384 Atoms (Note the number of atoms along each natural direction of␣
↪the bulk is (16, 16, 16))
Cohesive energy: -3.8281812272033404 eV/Atom

Total Volume: 276148.8097280001 Angstroms^3
Volume per atom: 16.854785750000005 Angstroms^3/Atom

Stress tensor:
[[1.29870561e-03 1.29377728e-19 1.68652768e-19]
 [1.29377728e-19 1.29870561e-03 1.29962170e-19]
 [1.68652768e-19 1.29962170e-19 1.29870561e-03]]

Bulk Modulus: 218.1717839967987 GPa
```

We can see how well the lattice constant and been modelled using a energy vs volume plot given by the `EOS` module in `ASE`



See Equation of state (EOS) in ASE[26] for more information.

4.6.2 2D and 3D Systems containing two lattice constant

1) Performing a broad overview of lattice constants

Like in the one lattice constant case, it is often useful to perform a broad overview of lattice constants to determine what lattice constants to focus on. For example, for determining the lattice constant of a Au hexagonal closed packed (HCP) lattice using the RGL potential, with parameters from Baletto *et al.* (DOI: 10.1063/1.1448484[27]). Here, we first look at lattice constants for `c` between 2.0 Å and 5.0 Å in increments of 0.1 Å, and `a` between 3.0 Å and 6.0 Å in increments of 0.1 Å. This is performed with the following `Run_LatticeFinder.py` script:

Listing 8: Run_LatticeFinder.py

```
from LatticeFinder import LatticeFinder_Program

symbol = 'Au'
lattice_type = 'HexagonalClosedPacked'
```

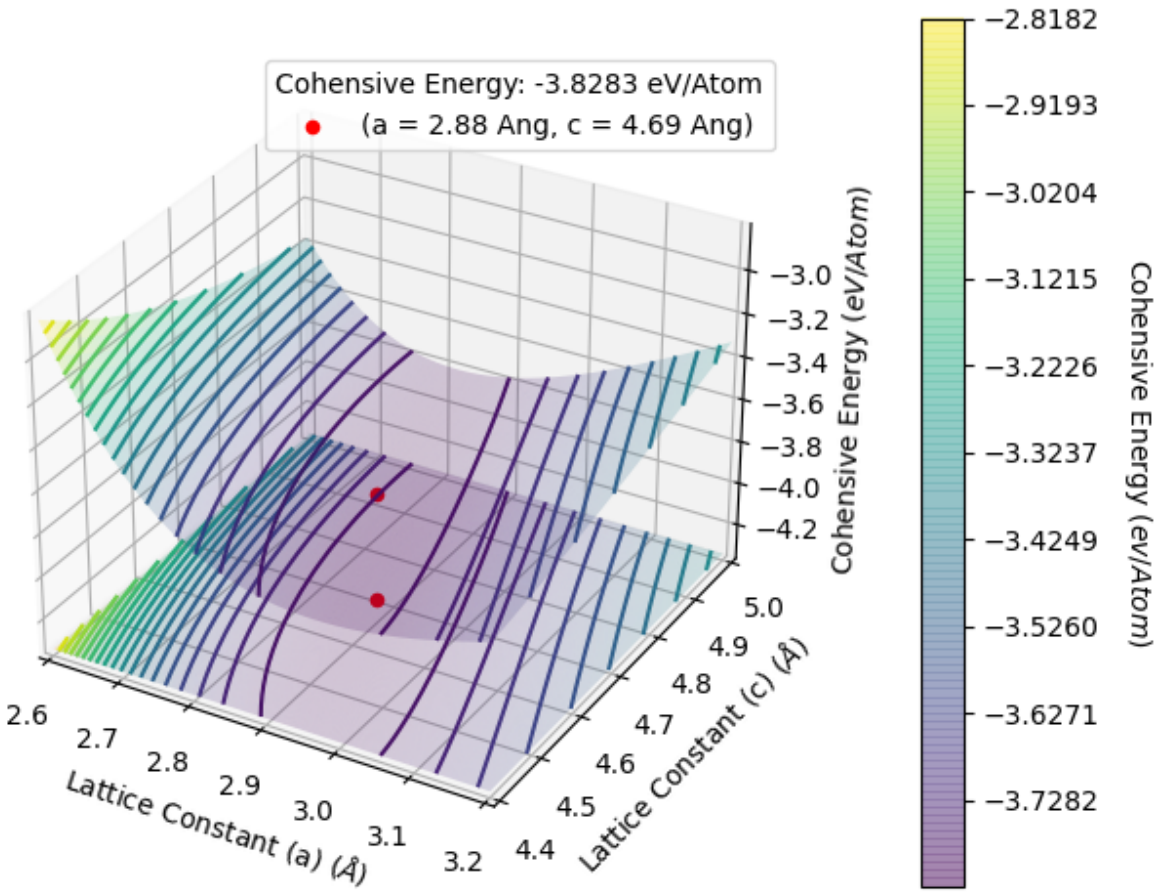
(continues on next page)

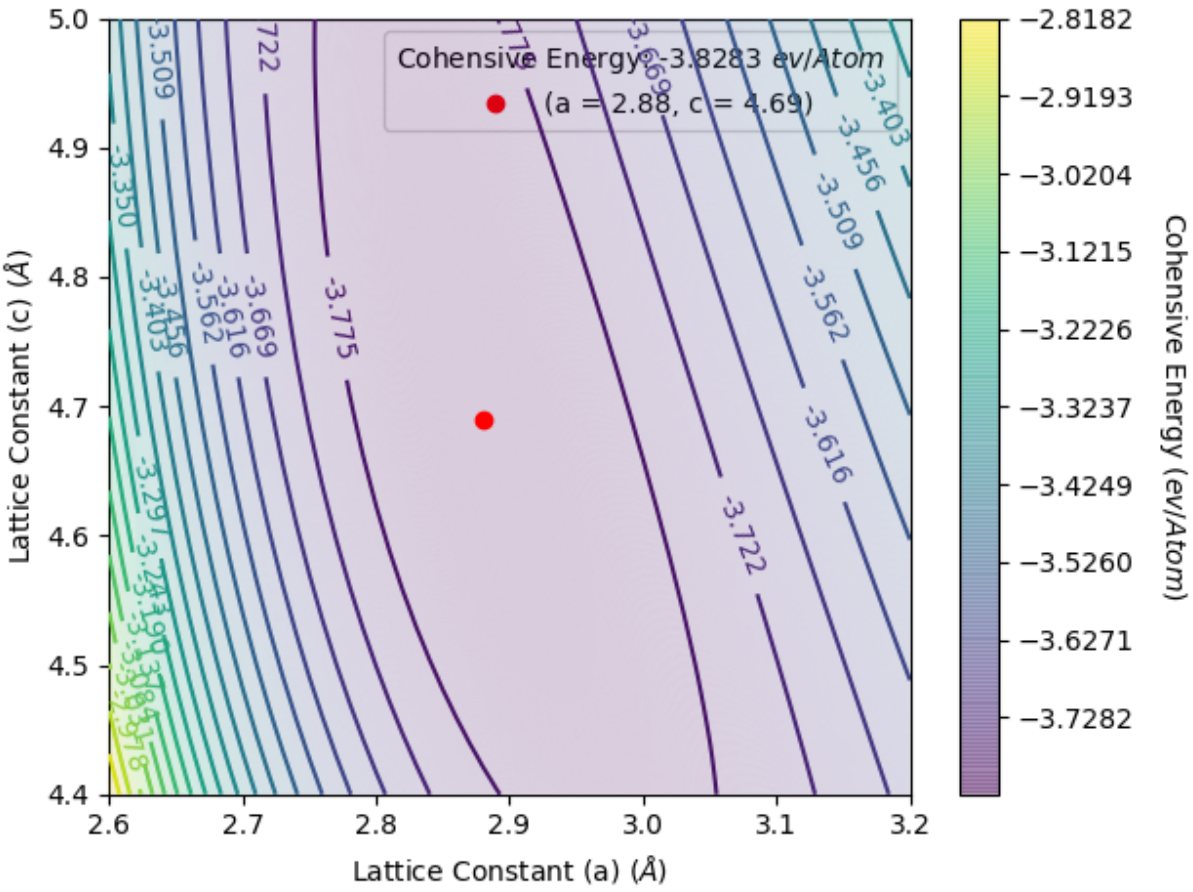
[26] https://wiki.fysik.dtu.dk/ase/tutorials/eos/eos.html
[27] https://doi.org/10.1063/1.1448484

(continued from previous page)

```

lattice_constant_parameters = {'a': (2.0,5.0,0.1), 'c': (3.0,6.0,0.1)}

from asap3.Internal.BuiltinPotentials import Gupta
# Parameter sequence: [p, q, a, xi, r0]
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
cutoff = 8
calculator = Gupta(Au_parameters, cutoff=cutoff, debug=False)

size_single = 28
size=(size_single,size_single,size_single)

directions=[]
miller=[]

limits = {'a': (2.6,3.2), 'c': (4.4,5.0)}
no_of_cpus = 8


LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, limits=limits, no_of_cpus=no_of_
↪cpus)
```

This gives an energy per atom vs lattice constant plots as below. We have use the `limits` section to zoom in on the important area of the potential energy surface to make it easier to see the bottom of the well.


Cohensive Energy: -3.8283 eV/Atom
(a = 2.88 Ang, c = 4.69 Ang)
Cohensive Energy (eV/Atom)
−3.0
−3.2
−3.4
−3.6
−3.8
−4.0
−4.2
Lattice Constant (a) (Å)
2.6
2.7
2.8
2.9
3.0
3.1
3.2
Lattice Constant (c) (Å)
5.0
4.9
4.8
4.7
4.6
4.5
4.4
Cohensive Energy (ev/Atom)
−2.8182
−2.9193
−3.0204
−3.1215
−3.2226
−3.3237
−3.4249
−3.5260
−3.6271
−3.7282



From this plot, we see that the minimum centres about a = 2.9 Å and c = 4.7 Å. Therefore, we will want to next perform a comprehensive search of the optimum lattice constant for a between 2.8 Å and 3.0 Å and c between 4.6 Å and 4.8 Å, at the least. This is because we set this script to scan the lattic constants in increments of 0.1 Å, therefore the precise lattice constant will lie within the a = 2.9 ± 0.1 Å and c = 4.7 ± 0.1 Å ranges.

2) Performing a comprehensive scan of lattice constants across a small range

We would like to perform a indepth scan of lattice constants between for a between 2.8 Å and 3.0 Å and c between 4.6 Å and 4.8 Å in increments of 0.001 Å. Unlike for LatticeFinder calculation for systems with one lattice constant, you can not add data easily to the plots made for systems with two lattice constants and therefore it is best to perform a new LatticeFinder calculation for your system. The `Run_LatticeFinder.py` file for running this is shown below:

Listing 9: Run_LatticeFinder.py

```
from LatticeFinder import LatticeFinder_Program

symbol = 'Au'
lattice_type = 'HexagonalClosedPacked'

lattice_constant_parameters = {'a': (2.8,3.0,0.001), 'c': (4.6,4.8,0.001)}

from asap3.Internal.BuiltinPotentials import Gupta
# Parameter sequence: [p, q, a, xi, r0]
```

(continues on next page)

(continued from previous page)

```
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
cutoff = 8
calculator = Gupta(Au_parameters, cutoff=cutoff, debug=False)

size_single = 28
size=(size_single,size_single,size_single)

directions=[]
miller=[]

limits = {'a': (2.6,3.2), 'c': (4.4,5.0)}
no_of_cpus = 8


LatticeFinder_Program(symbol, lattice_type, lattice_constant_parameters, calculator,␣
↪size=size, directions=directions, miller=miller, limits=limits, no_of_cpus=no_of_
↪cpus)
```

You can change the `lattice_constant_parameters` input from

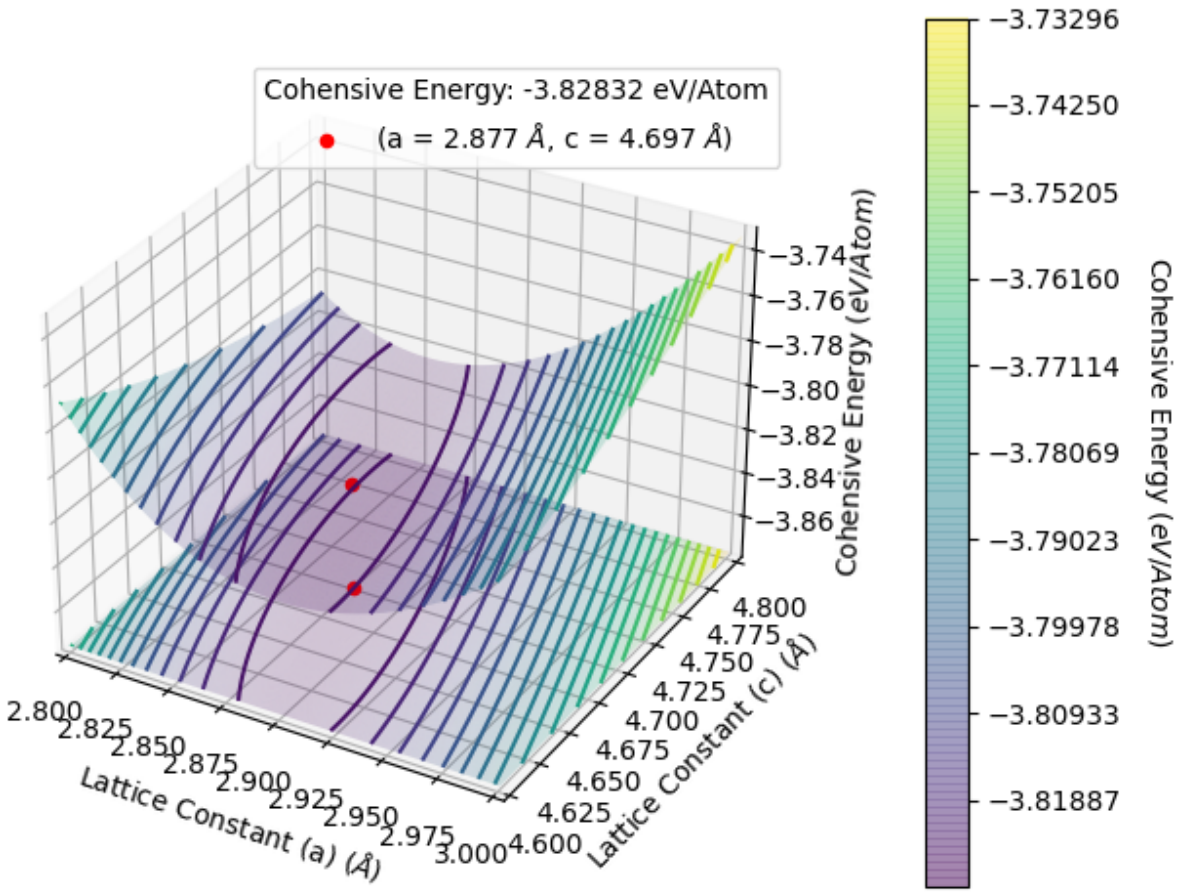
```
lattice_constant_parameters = {'a': (2.0,5.0,0.1), 'c': (3.0,6.0,0.1)}
```

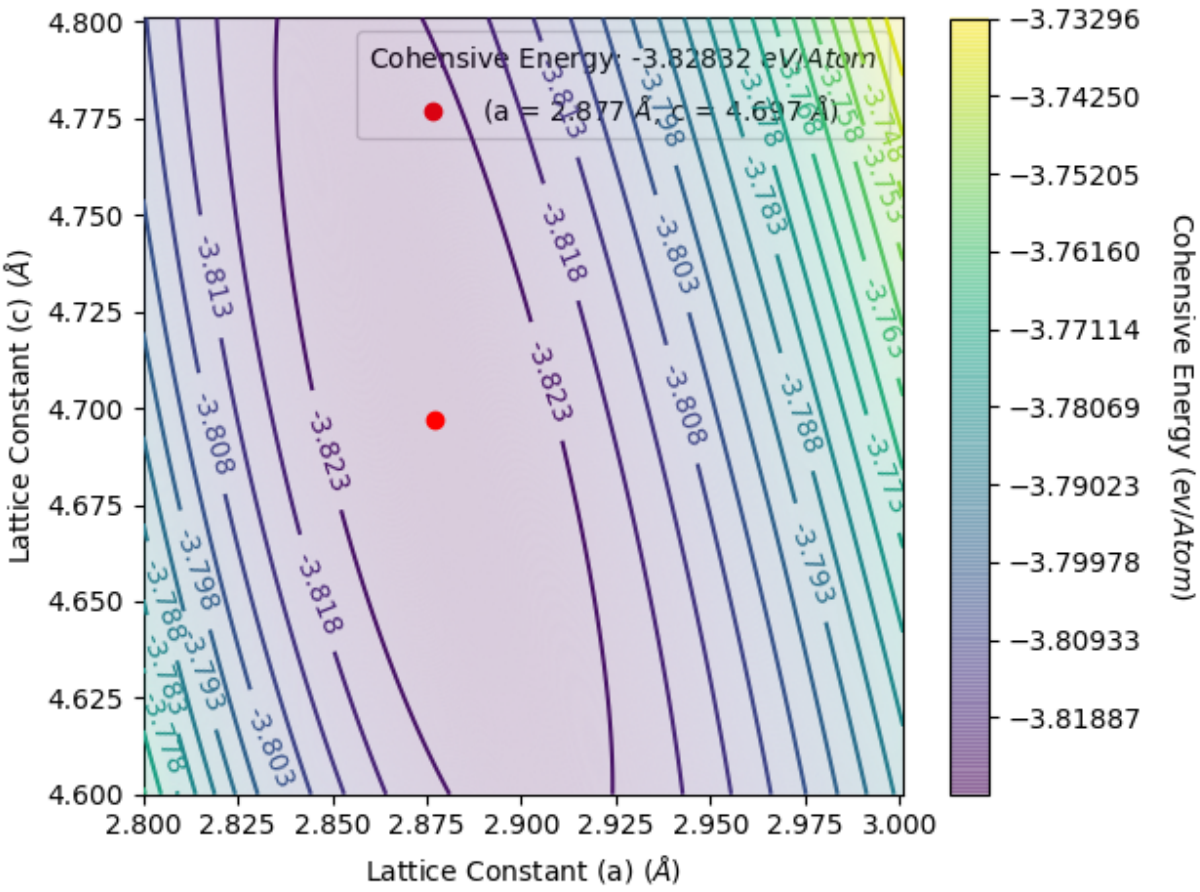
to

```
lattice_constant_parameters = {'a': (2.8,3.0,0.001), 'c': (4.6,4.8,0.001)}
```

and rerun LatticeFinder. LatticeFinder will perform calculations with lattice constants that you have not already obtained. Once you rerun LatticeFinder on your script again, you will get the following energy per atom vs lattice constant plot as below:

This gives an energy per atom vs lattice constant plots as below:


Cohensive Energy: -3.82832 eV/Atom
(a = 2.877 Å, c = 4.697 Å)
Cohensive Energy (eV/Atom)
−3.74
−3.76
−3.78
−3.80
−3.82
−3.84
−3.86
Lattice Constant (a) (Å)
2.800
2.825
2.850
2.875
2.900
2.925
2.950
2.975
3.000
Lattice Constant (c) (Å)
4.800
4.775
4.750
4.725
4.700
4.675
4.650
4.625
4.600
Cohensive Energy (eV/Atom)
−3.73296
−3.74250
−3.75205
−3.76160
−3.77114
−3.78069
−3.79023
−3.79978
−3.80933
−3.81887



The data from this is shown in `results_file.txt`

4.7 Examples of Running LatticeFinder with *Run_LatticeFinder.py*

[28] Provided in the LatticeFinder Github repository are examples of *Run_LatticeFinder.py*. Find these various examples at https://github.com/GardenGroupUO/LatticeFinder/tree/main/Examples.

We have also developed a Jupyter notebook with some examples of various *Run_LatticeFinder.py* that you can play with and muck around with. The Github repository for this Jupyter notebook can also be found at https://github.com/GardenGroupUO/LatticeFinder.

Along with this Jupyter notebook, we have also implemented this Jupyter notebook into Binder. Binder (https://mybinder.org/) is an interactive online platform that allows you to use Jupyter notebooks on an web browser without having to set up anything. It does all the setting up on a virtual computer for you. If you want to play around with the LatticeFinder program before you download it on your computer or if you need help when things go wrong using LatticeFinder on your computer, Binder+Jupyter is the best way to do this. **It is recommended that you try out the LatticeFinder program on Binder if you are interested or intending on using the LatticeFinder program**.

The Binder webpage can be found at:

[29] This will load a Binder page that will allow you to play about with LatticeFinder interactively in Binder. This Binder page may load quickly, or it may take 1 to 2 minutes to load. Don't refresh the page as Binder takes a good amount of

[28] https://mybinder.org/v2/gh/GardenGroupUO/LatticeFinder/main?urlpath=lab
[29] https://mybinder.org/v2/gh/GardenGroupUO/LatticeFinder/main?urlpath=lab

time to load. Get a coffee or a cup of tea while you wait.

Once this is done you will see a Jupyter notebook that you can interact with. Mess around with it as much as you want!

4.8 Helpful Programs to run LatticeFinder

LatticeFinder contains subsidary programs that you may find useful, especially for using LatticeFinder with VASP. In this article, we will introduce all of the programs that can be used with LatticeFinder. These programs can be run by typing the program you want to run into the terminal from whatever directory you are in.

The scripts and programs that we will be mentioned here are:

- *What to make sure is done before running any of these scripts.*
- *`Run_LatticeFinder_submitSL_slurm.py` - How to execute all VASP jobs individually as single jobs on slurm for lattice constant calculations*
- *`Run_overall_LatticeFinder_submitSL_slurm.py` - How to execute all your VASP jobs that have been collected together as packets for submission to slurm*
- *`LatticeFinder_Tidy_Finished_Jobs.py` - How to ...*

4.8.1 What to make sure is done before running any of these scripts.

If you installed LatticeFinder through pip3

If you installed the LatticeFinder program with `pip3`, these scripts will be installed in your bin. You do not need to add anything into your `~/.bashrc`. You are all good to go.

If you performed a Manual installation

If you have manually added this program to your computer (such as cloning this program from Github), you will need to make sure that you have included the `Subsidiary_Programs` folder into your `PATH` in your `~/.bashrc` file. All of these program can be found in the `Subsidiary_Programs` folder. To execute these programs from the `Subsidiary_Programs` folder, you must include the following in your `~/.bashrc`:

```
export PATH_TO_LatticeFinder="<Path_to_LatticeFinder>"
```

where `<Path_to_LatticeFinder>"` is the path to get to the genetic algorithm program. Also include somewhere before this in your `~/.bashrc`:

```
export PATH="$PATH_TO_LatticeFinder"/LatticeFinder/Subsidiary_Programs:$PATH
```

See more about this in *Installation of LatticeFinder*.

4.8.2 `Run_LatticeFinder_submitSL_slurm.py` - How to execute all VASP jobs individually as single jobs on slurm for lattice constant calculations

4.8.3 `Run_overall_LatticeFinder_submitSL_slurm.py` - How to execute all your VASP jobs that have been collected together as packets for submission to slurm

4.8.4 `LatticeFinder_Tidy_Finished_Jobs.py` - How to ...

4.9 Index

4.10 Python Module Index

CHAPTER

FIVE

INDICES AND TABLES

- *Index*
- *Python Module Index*