The Nanoclusters Interpolation Scheme Program (NISP) Documentation

Release 1

Geoffrey Weal

Aug 11, 2021

CONTENTS

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

Section author: Dr. Andreas Pedersen

Section author: Prof. Hannes Jónsson <hj@hi.is>

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

Page to cite with work from: A. L. Garden, A. Pedersen, H. Jónsson, "Reassignment of 'magic numbers' of decahedral and FCC structural motifs", Nanoscale, 10, 5124-5132 (2018), DOI: 10.1039/C7NR09440J[8]

[8] https://doi.org/10.1039/C7NR09440J

CHAPTER

ONE

WHAT IS THIS DOCUMENTATION ABOUT?

This documentation is designed to guide the user to use the Nanocluster Interpolation Scheme Program (NISP) program.

CHAPTER

TWO

WHAT IS NISP

The NISP program is an interpolation scheme that is designed to give an approximate guide for the estimated energies of unsymmetric nanoclusters based on energetic trends between perfect, closed-shell nanoclusters. This program will create all the perfect, close shell icosahedral, decahedral, and octahedral clusters that can be created between 13 atoms and an upper atom number limit. These nanoclusters are locally optimised using either an ASE, an ASE-integrated calculator, or with VASP. After the nanoclusters are locally optimised, the delta energy is obtained for each nanocluster before providing plots and text files that indicate the estimated energies of perfect, closed-shell and unsymmetric nanoclusters and how to remove atoms from the larger perfect closed-shell nanocluster to give unsymmetric nanoclusters with a certain number of atoms. See "Reassignment of 'magic numbers' of decahedral and FCC structural motifs" (DOI: 10.1039/C7NR09440J[9]) for more information about how this interpolation scheme works.

The algorithm was designed by Dr Anna Garden of the University of Otago, Dunedin, New Zealand, and Dr. Andreas Pedersen and Prof. Hannes Jónsson of the University of Iceland. The Github page for this program can be found at github.com/GardenGroupUO/NISP[10].

Dr. Anna Garden: blogs.otago.ac.nz/annagarden[11]

Dr. Andreas Pedersen: https://dk.linkedin.com/in/andreas-pedersen-a847025

Prof. Hannes Jónsson: english.hi.is/staff/hj[12]

[9] https://doi.org/10.1039/C7NR09440J
[10] https://github.com/GardenGroupUO/NISP
[11] https://blogs.otago.ac.nz/annagarden/
[12] https://english.hi.is/staff/hj

CHAPTER

THREE

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

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

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

[13]

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

CHAPTER

FOUR

INSTALLATION

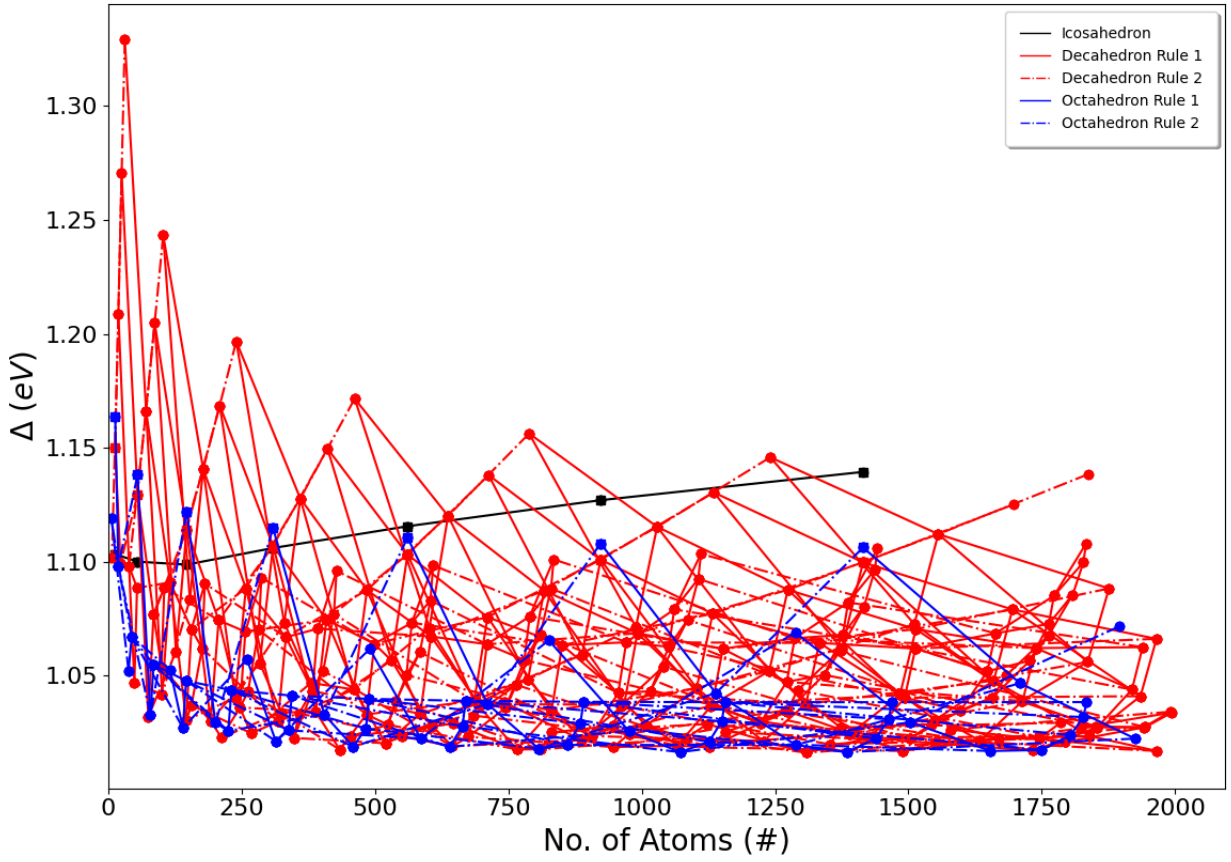
It is recommended to read the installation page before using the NISP program. See *Installation: Setting Up NISP and Pre-Requisites Packages* for more information. Note that you can install NISP through `pip3` and `conda`.

CHAPTER

FIVE

OUTPUT FILES THAT ARE CREATED BY NISP

An example of the plots that are created is the interpolation scheme plot, which shows all the estimated energies of nanoclusters across the size range of nanoclusters that you are measuring across. An example of this for Au nanoclusters, using the RGL potetial with parameters from Baletto *et al.* (DOI: 10.1063/1.1448484[14]), is shown below:



There are also other plots created by NIPS as well as other text documents that contain the delta energies of the various nanoclusters that you calculated, as well as instructions about how to remove atoms from certain nanoclusters in order to get icosahedral, decahedral, and octahedral nanoclusters with the particular number of atoms that you desire. Click here to see examples of all of these plots and text files[15].

[14] https://doi.org/10.1063/1.1448484
[15] https://github.com/GardenGroupUO/NISP/tree/main/Documentation/source/results

CHAPTER

SIX

TABLE OF CONTENTS

6.1 How the Nanocluster Interpolation Scheme Program (NISP) works

The NISP program is an interpolation scheme that is designed to give an approximate guide for the estimated energies of unsymmetric nanoclusters based on energetic trends between perfect, closed-shell nanoclusters.

This program will create all the perfect, close shell icosahedral, decahedral, and octahedral clusters that can be created between 13 atoms and an upper atom number limit. These nanoclusters are locally optimised using either an ASE, an ASE-integrated calculator, or with VASP.

After the nanoclusters are locally optimised, the delta energy is obtained for each nanocluster before providing plots and text files that indicate the estimated energies of perfect, closed-shell and unsymmetric nanoclusters and how to remove atoms from the larger perfect closed-shell nanocluster to give unsymmetric nanoclusters with a certain number of atoms.

See "Reassignment of 'magic numbers' of decahedral and FCC structural motifs" (DOI: 10.1039/C7NR09440J[16]) for more information about how this interpolation scheme works.

6.2 Installation: Setting Up NISP and Pre-Requisites Packages

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

6.2.1 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
```

[16] https://doi.org/10.1039/C7NR09440J

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
----------------------------- ---------
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[17]

Atomic Simulation Environment

NISP uses the atomic simulation environment (ASE) to construct the various types of icosahedral, decahedral, and octahedral nanoclusters that are used to perform the interpolation scheme. This allows LatticeFinder 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, and the local optimisers available to optimise nanoclusters created with NISP. Furthermore, ASE also offers useful tools for viewing, manipulating, reading and saving clusters and chemcial systems easily. Read more about ASE here[18]. 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[19], 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:

[17] https://pip.pypa.io/en/stable/installing/
[18] https://wiki.fysik.dtu.dk/ase/
[19] 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.

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

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
```

6.2.2 Setting up NISP

There are two ways to install NISP on your system. These ways are described below:

Install NISP through `pip3`

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

```
pip3 install --upgrade --user NISP
```

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

[20]

Install Organisms through `conda`

You can also install Organisms through `conda`, however I am not as versed on this as using `pip3`. See docs.conda.io[21] to see more information about this. Once you have installed anaconda on your computer, I believe you install NISP using `conda` by performing the following in your terminal.

```
conda install ase
conda install nisp
```

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

[22]

[20] https://pypi.org/project/NISP/
[21] https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html
[22] https://anaconda.org/GardenGroupUO/nisp

Manual installation

First, download NISP 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_Organisms_TO_LIVE_ON_YOUR_COMPUTER
git clone https://github.com/GardenGroupUO/NISP
```

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 Organisms program into `pwd`

```
pwd
```

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

```
export PATH_TO_NISP="<Path_to_NISP>"
export PYTHONPATH="$PATH_TO_NISP":$PYTHONPATH
```

where `"<Path_to_NISP>"` is the directory path that you place NISP (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.

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

```
export PATH="$PATH_TO_NISP"/NISP/Subsidiary_Programs:$PATH
```

See *Helpful Programs to run NISP* for more information about the programs that are available in the `Subsidiary_Programs` folder.

Other Useful things to know before you start

You may use squeue to figure out what jobs are running in slurm. For monitoring what genetic algorithm jobs are running, I have found the following alias useful. Include the following in your `~/.bashrc` (see How to execute all Trials using the JobArray Slurm Job Submission Scheme for what is going on in the below line)

```
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 Organisms program if you manually installed the Organisms

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

```
#########################################################
# Paths and Pythonpaths for NISP

export PATH_TO_NISP="<Path_to_NISP>"
export PYTHONPATH="$PATH_TO_NISP":$PYTHONPATH

export PATH="$PATH_TO_NISP"/NISP/Subsidiary_Programs:$PATH

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

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

6.3 *Interpolation_Script.py* - How to run NISP

In this article, we will look at how to run NISP. NISP is run through the `Interpolation_Script.py` python script. You can find examples of `Run.py` files at github.com/GardenGroupUO/NISP[23] under `Examples`. Also, you can try out this program by running an example script through a Jupyter notebook. See *Examples of running NISP* to get access to examples of running NISP through this Jupyter notebook!

6.3.1 Running the `Interpolation_Script.py` script

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

Listing 1: Interpolation_Script.py

```
1  from NISP import Run_Interpolation_Scheme
2  from RunMinimisation_Au import Minimisation_Function
3
4  input_information = {}
5  input_information['Element Type'] = 'Au'
6  input_information['Cohesive Energy'] = -3.82819360826 #-3.82819360826
7  input_information['Maximum No. of Atoms'] = 2000
8  input_information['Local Optimiser'] = Minimisation_Function
9
10 output_information = {}
11 output_information['Plot upper No of atom limit']   = None
12 output_information['Plot lower No of atom limit']   = None
13 output_information['Plot upper delta energy limit'] = None
14 output_information['Plot lower delta energy limit'] = None
15 output_information['Sizes to obtain instructions to create clusters for'] = [561,742,
   ↪923]#[37,38,44,55,147,40,888,1399]
16
17 no_of_cpus = 4
18 filename_prefix = ''
19
20 Run_Interpolation_Scheme(input_information=input_information,output_
   ↪information=output_information,no_of_cpus=no_of_cpus,filename_prefix=filename_
   ↪prefix)
```

(continues on next page)

[23] https://github.com/GardenGroupUO/NISP

(continued from previous page)

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

1) Input information for the interpolation scheme

We first load the information required by the interpolation scheme. All this information is loaded as entries into the dictionary called `input_information`.

The pieces of information required in `input_information` are:

- **Element Type** (*str.*): This is the type of element that the cluster is made up of.
- **Cohesive Energy** (*float*): This is the cohesive energy of the element you are using. See *How to obtain cohesive energies* to find about about how to obtain cohesive energies.
- **Maximum No. of Atoms** (*int*): The number of offspring generated per generation.
- **Local Optimiser** (*def/str.*): This is a local optimisation method that you will locally optimise clusters with as well as their delta energies. See *RunMinimisation.py - Writing a Local Minimisation Function for NISP* for information about the local optimiser works and is written.
  - You can also use **VASP to perform DFT local optimisations** on your clusters. Do this by setting the `'Local Optimiser'` in `input_information` as `input_information['Local Optimiser'] = 'VASP'`. See *How to perform NISP with VASP calculations* to learn more about how to perform VASP calculations on clusters created using NISP.
  - You can also select to **manually enter in the energies of the clusters**. To do this, enter in `input_information['Local Optimiser'] = 'Manual Mode'`. See *How to manually enter energy results into NISP* for more information about how to manually enter in energies for clusters into NISP.

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

```
input_information = {}
input_information['Element Type'] = 'Au'
input_information['Cohesive Energy'] = -3.82819360826 #-3.82819360826
input_information['Maximum No. of Atoms'] = 2000
input_information['Local Optimiser'] = Minimisation_Function
```

2) Output information for the interpolation scheme

We then load the information required by the interpolation scheme to plot the results from the interpolation scheme. The sizes of all the clusters that you would like to obtain possible clusters for are also inputted here and given as txt files.

All this information is loaded as entries into the dictionary called `output_information`.

The pieces of information required in `output_information` are:

- **Plot upper No of atom limit** (*int*): This is the upper size range that you would like to plot.
- **Plot lower No of atom limit** (*int*): This is the lower size range that you would like to plot.
- **Plot upper delta energy limit** (*float*): This is the upper delta energy range that you would like to plot.
- **Plot lower delta energy limit** (*float*): This is the lower delta energy range that you would like to plot.

- **Sizes to obtain instructions to create clusters for** (*list of ints*): These are all the sizes of clusters that you would like to obtain possible clusters for, including perfect, open-shell, and close-shell clusters. NISP will include text files that will include how to make all the symmetric and unsymmetric icosahedral, decahedral, and octahedral cluster that contain a particular cluster size.
- **Filename Prefix** (*str.*): This is the prefix of the name that you want to give to files that are create by the NISP program. This does not need to be given, as there is a default prefix given. The default filename prefix includes the element of the cluster as well as the maximum no. of atoms that the program was run up to.

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

```
output_information = {}
output_information['Plot upper No of atom limit']   = None
output_information['Plot lower No of atom limit']   = None
output_information['Plot upper delta energy limit'] = None
output_information['Plot lower delta energy limit'] = None
output_information['Sizes to obtain instructions to create clusters for'] = [561,742,
↪923]#[37,38,44,55,147,40,888,1399]
```

3) The number of CPUs used by the program and the filename prefix of input and output files

NISP can run for a long time, especially if you have set **Maximum No. of Atoms** to over 1000 atoms. Therefore, it is possible to run this program for a while. Therefore, it is possible to parallelise this program so that it run a bit faster. This can be set by setting the `no_of_cpus` variable. `no_of_cpus` must be set to an int. The default value for the `no_of_cpus` variable is `1`.

Furthermore, you can also give a custom name to the input and output files that you make/are made. This is given in `filename_prefix`. However, you do not need to do this. If you dont want to have a custom filename, do not include `filename_prefix` in your script or set `filename_prefix = ''`.

An example of `no_of_cpus` in `Interpolation_Script.py` is given below:

```
no_of_cpus = 4
filename_prefix = ''
```

Run NISP!

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

```
Run_Interpolation_Scheme(input_information=input_information,output_
↪information=output_information,no_of_cpus=no_of_cpus,filename_prefix=filename_
↪prefix)
```

6.3.2 Output files that are created by NISP

The NISP program will create a number of plots and text documents when it is run. See *Examples of data that the NISP program gives* to see the types of plots and text documents that NISP will make.

6.4 *RunMinimisation.py* - Writing a Local Minimisation Function for NISP

In this article, we will look at how to write the local optimisation method for NISP.

6.4.1 What is the Minimisation_Function

The `Minimisation_Function` is a definition that perform local optimisations during NISP. This is used by NISP as a `def` (i.e. as a function). This means that, rather than a variable being passed into NISP, a function is passed into the algorithm.

The implementation of the local minimisation process into NISP has been designed to be as free as possible, so that the user can use whatever local optimisation algorithm or program they want to use. In general, this algorithm will import a cluster in an ASE format from NISP. The user can locally optimise it before sending it back to NISP again in the ASE format.

Because of this flexibility, it is possible to use any type of calculator from ASE, ASAP, GWAP, LAMMPS, etc. It is even possible for the user to design this to use with non-python user-interface based local optimisers. See *How to write the Minimisation_Function for non-ASE Implemented Calculator* for information on how to write a `Minimisation_Function` def to do this.

If you want to use VASP to perform local optimisation calculations, see *How to perform NISP with VASP calculations*. If you want to use another long running programming like Quantum Espresso, you will need to enter in cluster energies from the program you use into NISP manually. See *How to manually enter energy results into NISP* for more information.

In the following documentation we will describe how the `Minimisation_Function` method is designed in a `RunMinimisation.py` file, and how you can make your own. Examples of `RunMinimisation.py` files used in NISP runs can be found in github.com/GardenGroupUO/NISP[24] in the directory path `Examples` (this should be found in github.com/GardenGroupUO/NISP/tree/main/Examples[25]).

6.4.2 Where to write the Minimisation_Function

The `Minimisation_Function` can be written into the Run.py file. However, as a personal preference and also to make the code cleaner to read, write and use, I put it into another python file. This file I have called `RunMinimisation.py`. This does not need to be the name of this file. For example, I have named this file `RunMinimisation_AuPd.py` when I wanted to keep a record that this minimisation python file contained the Gupta parameters and code for locally minimising a cluster using the Gupta potential for a cluster containing Au and Pd atoms.

Furthermore, the def `Minimisation_Function` does not even need to be called `Minimisation_Function`. It could be called `TheGuptaFunction`, `the_local_minimisation_function`, or `The_Electric_Eel_Function`. Again, I have just always called it `Minimisation_Function` for simplicity and for ease when using different `Interpolation_Script.py` files with different `Minimisation_Function` codes.

[24] https://github.com/GardenGroupUO/NISP
[25] https://github.com/GardenGroupUO/NISP/tree/main/Examples

However, it is important that this code is referenced somehow in your `Interpolation_Script.py` script if you want to locally optimise clusters during the NISP program. The algorithm is imported into `Interpolation_Script.py` as follows (You can also see this in *Interpolation_Script.py - How to run NISP*):

```
# The RunMinimisation.py algorithm is one set by the user. It contain the def␣
↪Minimisation_Function
# That is used for local optimisations. This can be written in whatever way the user␣
↪wants to perform
# the local optimisations. This is meant to be as free as possible.
from RunMinimisation import Minimisation_Function
```

where, in the above code, `RunMinimisation` is the name of the file the local minimisation code is found in (This file is called `RunMinimisation.py`), and `Minimisation_Function` is the name of the function that is found in the `RunMinimisation.py`. If you do it like this, make sure that your `RunMinimisation.py` file is in the same folder as your Run.py file.

6.4.3 How to write the Minimisation_Function

The `Minimisation_Function` must be written with the following requirements:

- **cluster** (*ase.Atoms*): This is the unoptimised version of the cluster.

NOTE: The **collection** and **cluster_name** variables do not need to be used in your `RunMinimisation.py` script if you are using an ASE or ASE implemented calculator and local optimisator. This information may be useful if you want the cluster to be locally optimised using an external program that can not be easily used with python (for example with VASP, see *How to write the Minimisation_Function for non-ASE Implemented Calculator*).

returns:

- **cluster** (*ase.Atoms*) - This is the optimised version of the cluster.

An example of a RunMinimisation.py file for a Gupta potential involving only Au atoms is given below:

Listing 2: RunMinimisation.py

```
'''
RunMinimisation.py, GRW, 8/6/17

This script is designed to locally optimise clusters.
'''
from asap3.Internal.BuiltinPotentials import Gupta
from ase.optimize import FIRE
from ase.io import write

def Minimisation_Function(cluster):
    cluster.pbc = False
    # Perform the local optimisation method on the cluster.
    # Parameter sequence: [p, q, a, xi, r0]
    #Au_parameters = {'Au': [10.229, 4.0360, 0.2061, 1.7900, 2.884]}
    r0 = 4.07/(2.0 ** 0.5)
    Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
    Gupta_parameters = Au_parameters
    cutoff = 1000
    calculator = Gupta(Gupta_parameters, cutoff=cutoff, debug=False)
    cluster.set_calculator(calculator)
    original_cluster = cluster.copy()
    dyn = FIRE(cluster,logfile=None)
    converged = False
```

(continues on next page)

(continued from previous page)

```
    try:
        dyn.run(fmax=0.01,steps=5000)
        converged = dyn.converged()
        if not converged:
            cluster_name = 'issue_cluster.xyz'
            errorMessage = 'The optimisation of cluster ' + str(original_cluster) + ' did not optimise completely.\n'
            errorMessage += 'The cluster of issue before optimisation has been saved as: '+str(cluster_name)
            write(cluster_name,original_cluster)
            raise Exception(errorMessage)
    except Exception as exception_message:
        cluster_name = 'issue_cluster.xyz'
        errorMessage = 'The optimisation of cluster ' + str(original_cluster) + ' did not optimise completely.\n'
        errorMessage += 'The cluster of issue before optimisation has been saved as: '+str(cluster_name)+'\n'
        errorMessage += exception_message
        write(cluster_name,original_cluster)
        raise Exception(errorMessage)
    return cluster
```

We will explain the components of this example below:

Importing external code

To begin, you will need to import all the external files that you will need so that you have the descriptor of the potential you want to use, and the local optimiser that you would like to use. In this example, the Gupta potential is used as the descriptor for the potential, while `FIRE` is the local optimiser that will be used to locally optimise the cluster.

```
from asap3.Internal.BuiltinPotentials import Gupta
from ase.optimize import FIRE
from ase.io import write
```

Preparing the cluster

First, it is usually a good idea to tell ase if you want the calculator to calculate the cluster with periodic boundary conditions `pbc` or not. In the case of the Gupta potential, we will include the line `cluster.pbc = False` to make sure that there are no boundary conditions on upon the cluster, since we do not want this and the Gupta potential does not need this turned on. For your potential, you may want to include this, or not.

```
cluster.pbc = False
```

Preparing the Potential, and setting up the local optimiser.

We would like to set up the parameters needed for the descriptor of the potential, attach the descriptor as a calculator to the `cluster`, and set up the local optimiser. In this example, Gupta is called a calculator. It contains a description of the Gupta potential that can be used to calculate the energy of `cluster`. We do this in the line `cluster.set_calculator(Gupta(Gupta_parameters, cutoff=1000, debug=True))`. For more information on how this works, see Tutorial on Using Calculators in ASE[26].

The last line, `dyn = FIRE(cluster)`, sets up the local optimiser, `FIRE`, to be used to locally minimise the cluster `cluster` (see Tutorial on Structure Optimization in ASE[27]).

See below for a example:

```
# Perform the local optimisation method on the cluster.
# Parameter sequence: [p, q, a, xi, r0]
#Au_parameters = {'Au': [10.229, 4.0360, 0.2061, 1.7900, 2.884]}
r0 = 4.07/(2.0 ** 0.5)
Au_parameters = {'Au': [10.53, 4.30, 0.2197, 1.855, r0]} # Baletto
Gupta_parameters = Au_parameters
cutoff = 1000
calculator = Gupta(Gupta_parameters, cutoff=cutoff, debug=False)
cluster.set_calculator(calculator)
original_cluster = cluster.copy()
dyn = FIRE(cluster,logfile=None)
```

Executing the local optimiser

We would like to now get the definition to run a local optimisation. This is done by performing `dyn.run(fmax=0.01,steps=5000)`. However I have found that if something breaks for some reason during the optimisation, this can completely stop the genetic algorithm in its tracks, and cause it to finish with a fatal error. You may want this to happen so that you can address issues when they arise, but sometimes it is hard to continue to work when it keeps happening. If you would like, you can make sure the genetic algorithm does not fail entirely by adding a Error Handling block, as shown in the example below:

```
converged = False
try:
    dyn.run(fmax=0.01,steps=5000)
    converged = dyn.converged()
    if not converged:
        cluster_name = 'issue_cluster.xyz'
        errorMessage = 'The optimisation of cluster ' + str(original_cluster) + ' did not optimise completely.\n'
        errorMessage += 'The cluster of issue before optimisation has been saved as:'+str(cluster_name)
        write(cluster_name,original_cluster)
        raise Exception(errorMessage)
except Exception as exception_message:
    cluster_name = 'issue_cluster.xyz'
    errorMessage = 'The optimisation of cluster ' + str(original_cluster) + ' did not optimise completely.\n'
    errorMessage += 'The cluster of issue before optimisation has been saved as:'+str(cluster_name)+'\n'
    errorMessage += exception_message
```

(continues on next page)

[26] https://wiki.fysik.dtu.dk/ase/ase/calculators/calculators.html
[27] https://wiki.fysik.dtu.dk/ase/ase/optimize.html

(continued from previous page)

```
38      write(cluster_name,original_cluster)
39      raise Exception(errorMessage)
```

You can also see that I have placed an if statement to determine if the local optimsation actually converged. I have found that it is useful to include a way of noting if the optimisation was able to converge or not. See more about How to perform a local optimisation in ASE here[28], or refer to the manual of the local optimiser you are using for more information on how to do this.

Return the Optimised Cluster and Info

Remember to return the `cluster` to the genetic algorithm so that it can use this information, as well as the optimiser cluster, to proceed to explore the potential energy surface of the cluster you wish to explore.

```
40  return cluster
```

6.4.4 How to write the Minimisation_Function for a ASE Implemented Calculator

If the descriptor for the potential you would like to use is implemented in ASE, it is very easy to implement this into your Minimisation_Function definition. You can use the example of RunMinimisation.py above, where the only component you need to change is the set_calculator function used by Opt_cluster. This is the bit of the code above that looks like this:

```
Gupta_parameters = {'Cu': [10.960, 2.2780, 0.0855, 1.224, 2.556]}
cluster.set_calculator(Gupta(Gupta_parameters, cutoff=1000, debug=True))
```

Instead of this, you can include all the parameters that you need for your potential before the set_calculator line. For example:

```
Potential_Parameters = ...
cluster.set_calculator(Potential(Potential_Parameters))
```

Where `Potential` is the potential you would like to use, and `Potential_Parameters` are all the parameters that `Potential` needs to work. Please consult the manual of the potential you would like to use to learn how to use that potential.

6.4.5 How to write the Minimisation_Function for non-ASE Implemented Calculator

In the previous section of this page we have been performing a local optimisation using ASE implemented calculators. However, you may want to use a calculator to locally optimise your cluster. This may only be possible by allowing the program you wish to use to itself completely locally optimise the cluster. This is no issue for us! We just need to be careful to implement the local optimisation using your own program, and make sure that the RunMinimisation.py file is constructed as follows:

Input into `Minimisation_Function`:

- **cluster** (*ASE.Atoms*): This is the unoptimised version of the cluster.

returns:

- **cluster** (*ASE.Atoms*) - This is now the optimised version of the cluster.

A general script for locally optimising however you want to is given below:

[28] https://wiki.fysik.dtu.dk/ase/ase/optimize.html

Listing 3: RunMinimisation_General.py

```
'''
RunMinimisation.py, GRW, 8/6/17

This script is designed to locally optimise clusters.
'''
import time
from ase.io import write
from subprocess import Popen

def Minimisation_Function(cluster):
    cluster.pbc = What_you_want # make sure that the periodic boundry conditions are set off
    # ------------------------------------------------------
    # Perform any pre-optimisation work here
    # ------------------------------------------------------
    startTime = time.time(); converged = False
    try:
        Popen(['run','external','program'])
    except Exception as exception_message:
        cluster_name = 'issue_cluster.xyz'
        errorMessage = 'The optimisation of cluster ' + str(original_cluster) + ' did not optimise completely.\n'
        errorMessage += 'The cluster of issue before optimisation has been saved as: '+str(cluster_name)+'\n'
        errorMessage += exception_message
        write(cluster_name,original_cluster)
        raise Exception(errorMessage)
    endTime = time.time()
    # ------------------------------------------------------
    # Perform any post-optimisation work here
    # ------------------------------------------------------
    return cluster
```

If you want to locally optimise your clusters with computational heavy program, like VASP and QuantumEspresso, it may be better to run these programs manually and then enter the results of this into an input file. See *How to manually enter energy results into NISP* to obtain more information about how to manually enter energy results into NISP.

6.5 How to perform NISP with VASP calculations

In this article, we will look at how to run NISP where VASP is used to locally optimise nanoclusters. The `Interpolation_Script.py` python script that is used is the same as shown previously in *Interpolation_Script.py - How to run NISP*, but with an extra component. An example of a `Interpolation_Script.py` python script that uses VASP is shown below:

Listing 4: Interpolation_Script.py

```
from NISP import Run_Interpolation_Scheme

input_information = {}
input_information['Element Type'] = 'Au'
input_information['Cohesive Energy'] = -3.82819360826 #-3.82819360826
input_information['Maximum No. of Atoms'] = 400
```

(continues on next page)

(continued from previous page)

```
input_information['Local Optimiser'] = 'VASP'

slurm_information = {}
slurm_information['project'] = 'uoo00084'
slurm_information['time'] = '72:00:00'
slurm_information['nodes'] = 1
slurm_information['ntasks_per_node'] = 16
slurm_information['mem-per-cpu'] = '4G'
slurm_information['partition'] = 'large'
slurm_information['email'] = 'slurmnotifications@slurmnotifications.com'
slurm_information['vasp_version'] = 'VASP/5.4.4-intel-2017a'
slurm_information['vasp_execution'] = 'vasp_std'

input_information['Slurm Information'] = slurm_information

output_information = {}
output_information['Plot upper No of atom limit']    = None
output_information['Plot lower No of atom limit']    = None
output_information['Plot upper delta energy limit'] = None
output_information['Plot lower delta energy limit'] = None
output_information['Sizes to obtain instructions to create clusters for'] = [37,38,44,
↪55,147,40]

no_of_cpus = 1
filename_prefix = ''

Run_Interpolation_Scheme(input_information=input_information,output_
↪information=output_information,no_of_cpus=no_of_cpus,filename_prefix=filename_
↪prefix)
```

6.5.1 The `input_information` dictionary

The extra component has been included in the `input_information` dictionary is the `slurm_information`. `slurm_information` is a dictionary that contains all the information that is needed to create the `submit.sl` files required to submit VASP calculations to slurm. The following information is need 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[29] 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: `''`).

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

- **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'`).

Make sure that you include `'slurm_information'` in `input_information` by writing in `Interpolation_Script.py`

```
input_information['Slurm Information'] = slurm_information
```

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

Listing 5: Interpolation_Script.py

```
slurm_information = {}
slurm_information['project'] = 'uoo00084'
slurm_information['time'] = '72:00:00'
slurm_information['nodes'] = 1
slurm_information['ntasks_per_node'] = 16
slurm_information['mem-per-cpu'] = '4G'
slurm_information['partition'] = 'large'
slurm_information['email'] = 'slurmnotifications@slurmnotifications.com'
slurm_information['vasp_version'] = 'VASP/5.4.4-intel-2017a'
slurm_information['vasp_execution'] = 'vasp_std'

input_information['Slurm Information'] = slurm_information
```

6.5.2 Other files that you will need

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

- `INCAR`: This contains all the setting that are required by VASP to perform calculations.
- `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.

These files will be copied by NISP into each nanocluster folder. See an example of a setup of NISP for VASP here[30].

6.5.3 What to do after you have run NISP

After you run NISP, this will create a new folder called `VASP_Clusters`, which contains subfolders of all your nanoclusters. Each subfolder will contain a `POSCAR`, `INCAR`, `POTCAR`, `KPOINTS`, and `submit.sl` that are needed by VASP to perform DFT calculations. Each nanocluster is ready to be locally optimised with VASP.

You will find that there are many nanoclusters are created by NISP. To submit all of these nanoclusters for local minimisation by VASP, you can execute the program called `Run_submitSL_slurm.py` which will execute all of your nanocluster DFT optimisations to slurm. To run this script, type `Run_submitSL_slurm.py` into the terminal inside of your newly created `VASP_Clusters` folder. See *Installation of NISP* for how to use this `Run_submitSL_slurm.py` script.

[30] https://github.com/GardenGroupUO/NISP/tree/main/Examples/VASP

6.6 How to manually enter energy results into NISP

It is also possible to run NISP in manual mode, where NISP creates xyz files of all your nanoclusters, and a `Au_Max_Size_YYYY_atoms_interpolation_scheme_input_file.txt`, where `YYYY` is the maximum size cluster that you have chosen to measure up to. In this article, we will look at how to run NISP in manual mode. An example of a `Interpolation_Script.py` python script that is set to manual mode is shown below:

Listing 6: Interpolation_Script.py

```
from NISP import Run_Interpolation_Scheme

input_information = {}
input_information['Element Type'] = 'Au'
input_information['Cohesive Energy'] = -3.82819360826 #-3.82819360826
input_information['Maximum No. of Atoms'] = 2000
input_information['Local Optimiser'] = 'Manual Mode'

output_information = {}
output_information['Plot upper No of atom limit']   = None
output_information['Plot lower No of atom limit']   = None
output_information['Plot upper delta energy limit'] = None
output_information['Plot lower delta energy limit'] = None
output_information['Sizes to obtain instructions to create clusters for'] = [561,742,
↪923]#[37,38,44,55,147,40,888,1399]

no_of_cpus = 1
filename_prefix = ''

Run_Interpolation_Scheme(input_information=input_information,output_
↪information=output_information,no_of_cpus=no_of_cpus,filename_prefix=filename_
↪prefix)
```

Here, `input_information['Local Optimiser'] = 'Manual Mode'`

6.6.1 What to do after you have run NISP

Once you have run NISP, you will find that NISP will have made a folder called `Clusters` and a file called `Au_Max_Size_YYYY_atoms_interpolation_scheme_input_file.txt`, where `YYYY`.

- `Clusters`: This folder contains all the `xyz` files of the clusters that you need to obtain energies for.
- `Au_Max_Size_YYYY_atoms_interpolation_scheme_input_file.txt`: This is the file that you want to place all the energies for each nanocluster in this list.

6.6.2 What to do once you have got all the energies of your nanoclusters

Once you have obtained all the energies for each of the nanoclusters in the `Clusters` folder, you want to add these energies to the right most side of each column of the `Au_Max_Size_YYYY_atoms_interpolation_scheme_input_file.txt` column. For example, you want to place your energies into each place in the file below where _____ is given

```
Element: Au Max_Size: 2000
Enter the energies of the clusters below to the right most of each line (not the␣
↪delta energies, NISP can do that for you later)
------------------------------
```

(continues on next page)

(continued from previous page)

```
Icosahedron
13      2           _______
55      3           _______
147     4           _______
309     5           _______
561     6           _______
923     7           _______
1415    8           _______
Octahedron
6       (2, 0)      _______
19      (3, 0)      _______
13      (3, 1)      _______
44      (4, 0)      _______
38      (4, 1)      _______
85      (5, 0)      _______
79      (5, 1)      _______
55      (5, 2)      _______
146     (6, 0)      _______
...
Decahedron
7       (2, 1, 0)   _______
49      (2, 1, 1)   _______
156     (2, 1, 2)   _______
358     (2, 1, 3)   _______
685     (2, 1, 4)   _______
13      (2, 2, 0)   _______
75      (2, 2, 1)   _______
212     (2, 2, 2)   _______
454     (2, 2, 3)   _______
831     (2, 2, 4)   _______
1373    (2, 2, 5)   _______
19      (2, 3, 0)   _______
101     (2, 3, 1)   _______
268     (2, 3, 2)   _______
550     (2, 3, 3)   _______
977     (2, 3, 4)   _______
1579    (2, 3, 5)   _______
25      (2, 4, 0)   _______
127     (2, 4, 1)   _______
324     (2, 4, 2)   _______
646     (2, 4, 3)   _______
1123    (2, 4, 4)   _______
1785    (2, 4, 5)   _______
31      (2, 5, 0)   _______
153     (2, 5, 1)   _______
380     (2, 5, 2)   _______
742     (2, 5, 3)   _______
1269    (2, 5, 4)   _______
1991    (2, 5, 5)   _______
23      (3, 1, 0)   _______
100     (3, 1, 1)   _______
262     (3, 1, 2)   _______
539     (3, 1, 3)   _______
961     (3, 1, 4)   _______
39      (3, 2, 0)   _______
146     (3, 2, 1)   _______
348     (3, 2, 2)   _______
```

(continues on next page)

(continued from previous page)

```
675     (3, 2, 3)   _______
1157    (3, 2, 4)   _______
1824    (3, 2, 5)   _______
55      (3, 3, 0)   _______
192     (3, 3, 1)   _______
434     (3, 3, 2)   _______
811     (3, 3, 3)   _______
1353    (3, 3, 4)   _______
71      (3, 4, 0)   _______
238     (3, 4, 1)   _______
...
```

6.7 How to obtain cohesive energies

One of the pieces of information that is needed to obtain delta energies for this program is the cohesive energy for the crystal structure for the metal of interest. This is the energy for a crystal in a cell with periodic boundary conditions divided by the number of atoms in the cell (given as eV/atom).

We have developed a program designed to help you to obtain the cohesive energy called LatticeFinder. You can find LatticeFinder at github.com/GardenGroupUO/LatticeFinder and documentation at latticefinder.readthedocs.io

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:

[31] 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 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!

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

6.8 What are delta energies?

The delta energy is an estimate of the surface energy of a cluster. The delta energy is defined as:

$$\Delta = \frac{E - NE_{coh}}{N^{2/3}}$$

where Δ is the delta energy, E is the energy of the cluster, N is the number of atoms in the cluster, and E_{coh} is the cohesive energy of a crystal lattice of the same metal that the clusters contain (where E_{coh} is given in eV/atom). The $N^{2/3}$ term is an estimate of the number of surface atoms that the cluster contains in a spherical cluster. The units of Δ are eV/atom, or eV per estimate number of surface atoms.

See the article for more information about the delta energy: A. L. Garden, A. Pedersen, H. Jónsson, "Reassignment of 'magic numbers' of decahedral and FCC structural motifs", Nanoscale, 10, 5124-5132 (2018), DOI: 10.1039/C7NR09440J[32]

6.9 Examples of Running NISP with *Interpolation_Script.py*

[33] Provided in the NISP Github repository are examples of *Interpolation_Script.py* (and *RunMinimisation.py*, see *RunMinimisation.py - Writing a Local Minimisation Function for the Genetic Algorithm* for more information about this file) scripts that you can try out. Find these various examples at https://github.com/GardenGroupUO/NISP/tree/main/Examples.

We have also developed a Jupyter notebook with some examples of various *Interpolation_Script.py* (and *RunMinimisation.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/NISP.

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 NISP program before you download it on your computer or if you need help when things go wrong using NISP on your computer, Binder+Jupyter is the best way to do this. **It is recommended that you try out the NISP program on Binder if you are interested or intending on using the NISP program**.

The Binder webpage can be found at:

[34] This will load a Binder page that will allow you to play about with NISP 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 time to load. Get a coffee or a cup of tea while you wait.

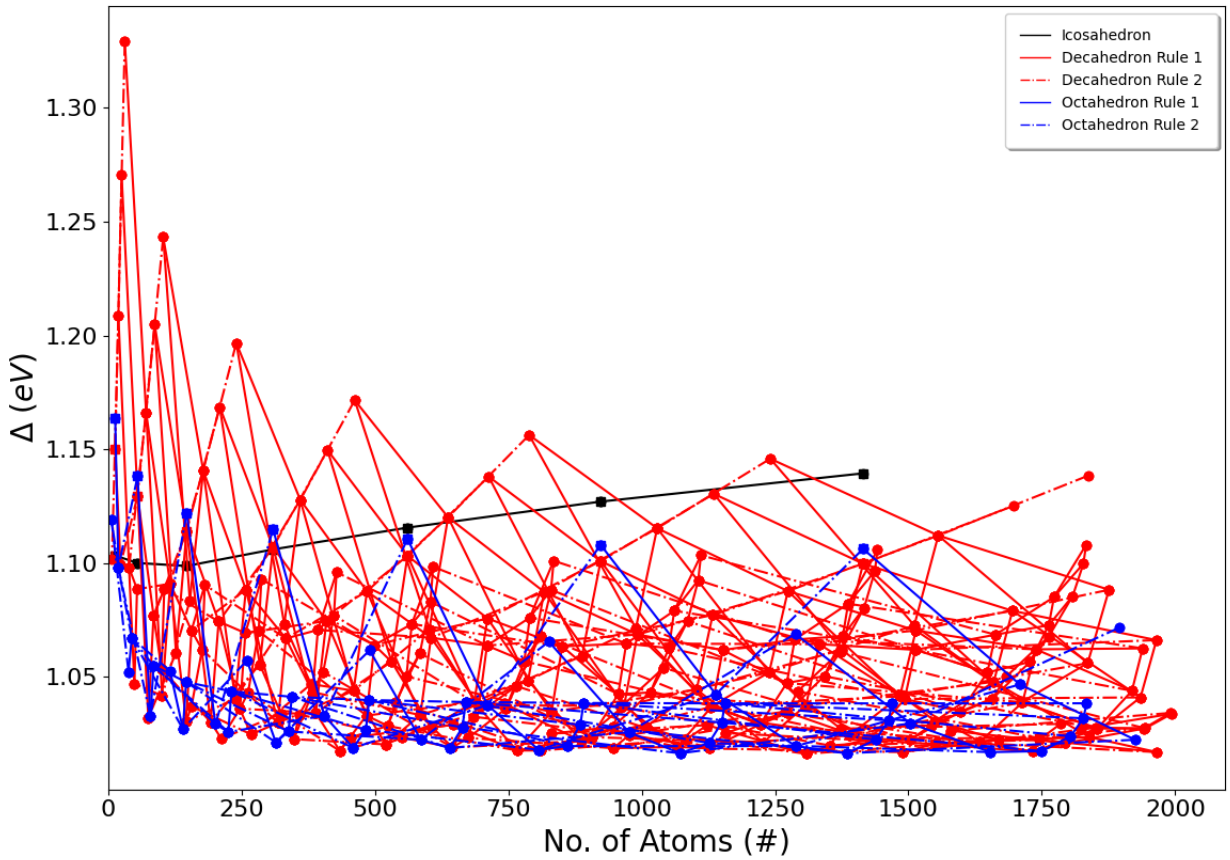
Once this is done you will see a Jupyter notebook that you can interact with. Mess around with it as much as you want!

[32] https://doi.org/10.1039/C7NR09440J
[33] https://mybinder.org/v2/gh/GardenGroupUO/NISP/main?urlpath=lab
[34] https://mybinder.org/v2/gh/GardenGroupUO/NISP/main?urlpath=lab

6.10 Examples of data that the NISP program gives

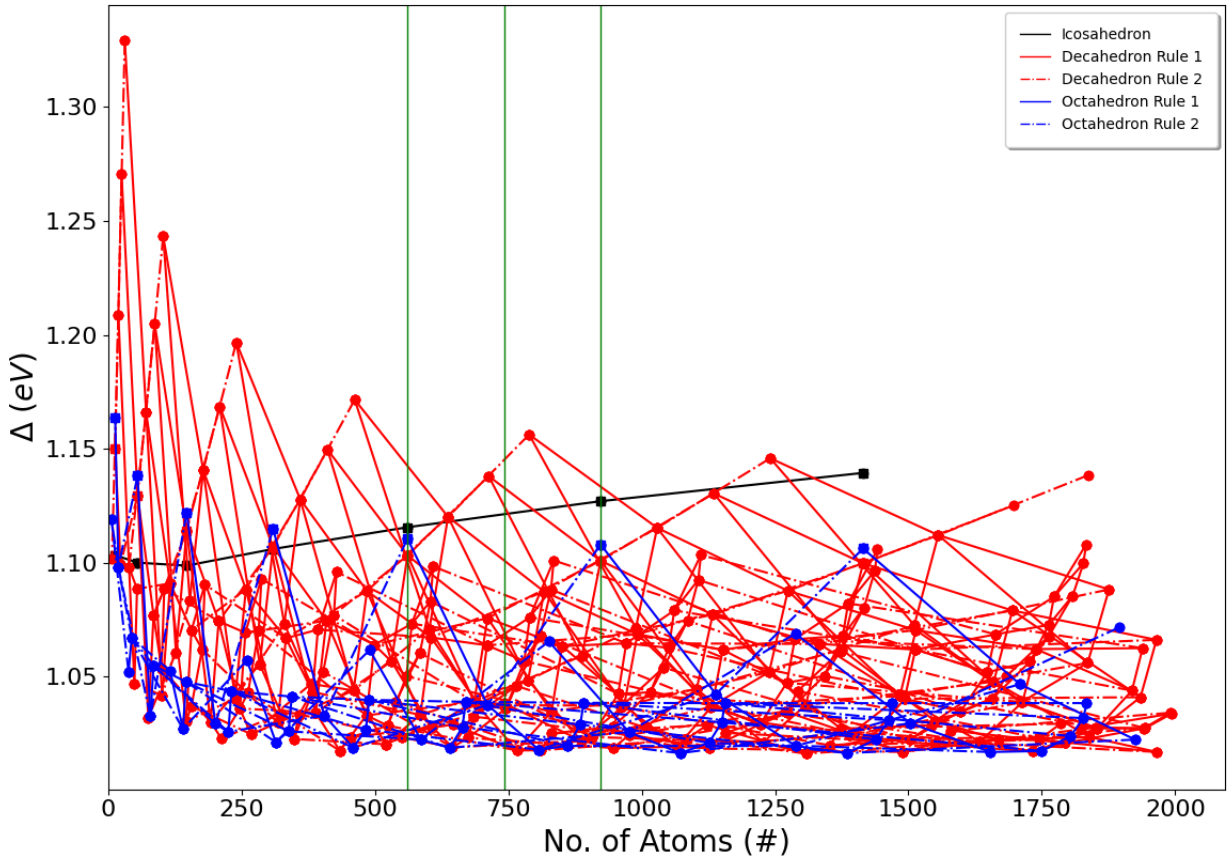
Once NISP has completed or once VASP is able to get or has been given all the energies of nanoclusters, NISP will give a set of plots and data. Examples of these can be found on the github page (Click here to see examples of all of these plots and text files[35]) and given below.

Firstly, NISP will output a series of plots and text files about various features of the nanoclusters. The first plot that is given is the interpolation scheme plot, which shows all the estimated energies of nanoclusters across the size range of nanoclusters that you are measuring across. An example of this for Au nanoclusters, using the RGL potetial with parameters from Baletto *et al.* DOI: 10.1063/1.1448484[36], is shown below:



The second plot is the same interpolation scheme plot shown above, but with lines through it at the places that you want to obtain cluster with the particular number of atoms that you desire.

[35] https://github.com/GardenGroupUO/NISP/tree/main/Documentation/source/results
[36] https://doi.org/10.1063/1.1448484



There are also a few text documents that are created by NISP. The first is a file called `XX_Max_Size_YYYY_atoms_interpolation_scheme_results_file.txt`, where `XX` is the element of interest and `YYYY` is the max cluster size that were generated. This file contains the number of atoms, the base structure, and the delta energy of clusters generated by NISP. An example for Au nanoclusters, using the RGL potetial with parameters from Baletto *et al.* (DOI: 10.1063/1.1448484[37]), is shown below:

```
Element: Au Max_Size: 2000
------------------------------
Icosahedron
13      2       1.1032754037994479
55      3       1.100003781794644
147     4       1.0987059970335966
309     5       1.1059920963404124
561     6       1.115553112131466
923     7       1.1269907548069968
1415    8       1.1394066237505638
Octahedron
6       2       0       1.1190094015786765
13      3       1       1.1638224621856568
19      3       0       1.0979832400015577
38      4       1       1.0521405814430245
44      4       0       1.067143817259378
55      5       2       1.138189866616928
79      5       1       1.0325428000659487
85      5       0       1.0548437019581978
```

[37] https://doi.org/10.1063/1.1448484

```
116     6       2       1.0524850917350583
140     6       1       1.0270074284856954
146     6       0       1.0476962360309887
147     7       3       1.1219838090199536
201     7       2       1.0292555823343328
225     7       1       1.0257371723399247
231     7       0       1.0436029688201243
260     8       3       1.057200064495081
309     9       4       1.1148330197406788
314     8       2       1.0212213249548565
338     8       1       1.0259936196321453
344     8       0       1.0412199350502054
405     9       3       1.0329243668424515
459     9       2       1.0187947176133691
483     9       1       1.0267388713457173
489     9       0       1.039780476733651
490     10      4       1.0616618424124262
561     11      5       1.1105816893340137
586     10      3       1.0223377246105532
640     10      2       1.0187179221666112
664     10      1       1.027788871978976
670     10      0       1.038919349769551
711     11      4       1.0375673369171623
807     11      3       1.0179067773491761
826     12      5       1.065451683596655
861     11      2       1.019557580472791
885     11      1       1.0288946740343523
891     11      0       1.0384954016447392
923     13      6       1.107986539327558
976     12      4       1.025654241565213
1072    12      3       1.0163941697802397
1126    12      2       1.0209225610011508
1139    13      5       1.0423080490997185
1150    12      1       1.0298879943900578
1156    12      0       1.0383295029773996
1288    14      6       1.0688693013913717
1289    13      4       1.0195493907284858
1385    13      3       1.0164262827288304
1415    15      7       1.1062004272998374
1439    13      2       1.0223727827352689
1463    13      1       1.0308977218222044
1469    13      0       1.03831689605
1504    14      5       1.0294836958637996
1654    14      4       1.0169058606393127
1709    15      6       1.0466394541790955
1750    14      3       1.0173524609832563
1804    14      2       1.0238887840747113
1828    14      1       1.0317899061516245
1834    14      0       1.0383656024767802
1896    16      7       1.0716861319137945
1925    15      5       1.0223405670750616
Decahedron
7       2       1       0       1.1017937744741588
13      2       2       0       1.1501771612388278
```

```
19      2       3       0       1.2088433973490311
23      3       1       0       1.0991995948493911
25      2       4       0       1.270445700676711
31      2       5       0       1.3291123029004166
39      3       2       0       1.0979830050918213
49      2       1       1       1.046982374072724
54      4       1       0       1.088495245255538
55      3       3       0       1.1296049882220227
71      3       4       0       1.1661887520089607
75      2       2       1       1.0319573159905462
85      4       2       0       1.076821815165658
87      3       5       0       1.204914273317384
100     3       1       1       1.0534128583165623
101     2       3       1       1.0416770307305245
103     3       6       0       1.2432615429056917
105     5       1       0       1.0886088712630677
116     4       3       0       1.0908967227610682
127     2       4       1       1.0605157146566482
146     3       2       1       1.0303203294892262
147     4       4       0       1.1140979959448842
153     2       5       1       1.0834524470461897
156     2       1       2       1.037045376222584
156     5       2       0       1.070347690927622
176     4       1       1       1.0620065432388022
178     4       5       0       1.1406902765651208
181     6       1       0       1.0903794185940532
192     3       3       1       1.0301186256260826
207     5       3       0       1.0742989944081809
209     4       6       0       1.1685437607098472
212     2       2       2       1.0228104604219803
238     3       4       1       1.0404010284411753
240     4       7       0       1.1966540041432234
247     4       2       1       1.036014358062041
257     6       2       0       1.0692388973011722
258     5       4       0       1.0883549664068048
262     3       1       2       1.0429429912267771
268     2       3       2       1.0246779196449565
282     5       1       1       1.0700982716052696
284     3       5       1       1.055472983804519
287     7       1       0       1.0929191650554706
309     5       5       0       1.1069784332433437
318     4       3       1       1.0297841036321091
324     2       4       2       1.0337818972196062
330     3       6       1       1.0730542403198535
333     6       3       0       1.0668426362975572
348     3       2       2       1.0226277438650595
358     2       1       3       1.0334378266783604
360     5       6       0       1.1277378513982543
380     2       5       2       1.0468060547106035
383     5       2       1       1.043254208992726
389     4       4       1       1.0341483142948795
393     7       2       0       1.0706031333849775
403     4       1       2       1.0521237254180393
409     6       4       0       1.074547548598091
```

411	5	7	0	1.149603597806501
423	6	1	1	1.077075334712185
428	8	1	0	1.09591162307455
434	3	3	2	1.0174940145425964
454	2	2	3	1.0233268355640692
460	4	5	1	1.0439022684287886
462	5	8	0	1.1715415984583404
484	5	3	1	1.0329704267131634
485	6	5	0	1.0875814949362173
499	7	3	0	1.0639323276004338
520	3	4	2	1.0202899991964027
524	4	2	2	1.0284304988216484
531	4	6	1	1.0566686062943933
539	3	1	3	1.0389246298086476
550	2	3	3	1.0233332663163037
559	6	2	1	1.050196914027839
561	6	6	0	1.1031598197987613
569	8	2	0	1.0732080357410643
584	5	1	2	1.0602254029875406
585	5	4	1	1.033189561449008
602	4	7	1	1.0709713062811868
604	7	1	1	1.0829999758805005
605	7	4	0	1.0671736748776157
606	3	5	2	1.0276807186955992
609	9	1	0	1.0983813564487561
637	6	7	0	1.1201040960474864
645	4	3	2	1.0189559203517278
646	2	4	3	1.0289286029460252
675	3	2	3	1.023158753407653
685	2	1	4	1.032425793681082
686	5	5	1	1.038910784777661
692	3	6	2	1.0379064353089364
695	6	3	1	1.0374501606276012
710	8	3	0	1.0637321705675284
711	7	5	0	1.0756558386813608
713	6	8	0	1.1379992874619629
742	2	5	3	1.03827731220149
745	5	2	2	1.0359104458171444
765	4	1	3	1.0459262974263792
766	4	4	2	1.017619332880287
780	7	2	1	1.0568265120815945
787	5	6	1	1.0483684202159638
789	6	9	0	1.156251396882655
790	9	2	0	1.0759816981316934
810	6	1	2	1.0681693221915103
811	3	3	3	1.0176348896347907
817	7	6	0	1.087348253000269
830	8	1	1	1.0882114367165607
831	2	2	4	1.0250367116601453
831	6	4	1	1.0345420345069662
835	10	1	0	1.1009748440852132
851	8	4	0	1.0633601935272483
887	4	5	2	1.0218086688043853
888	5	7	1	1.0591390340451714

```
906     5       3       2       1.023467445865111
923     7       7       0       1.1008672965324509
946     4       2       3       1.0279883528038825
947     3       4       3       1.018644613574685
956     7       3       1       1.0423911448167364
961     3       1       4       1.0367299241904282
967     6       5       1       1.0372599369793827
971     9       3       0       1.0646114378370695
977     2       3       4       1.0245090342096572
989     5       8       1       1.0713361628703204
992     8       5       0       1.068570954186557
1008    4       6       2       1.0280212280594143
1016    6       2       2       1.0430968673595469
1029    7       8       0       1.1152778795348373
1041    5       1       3       1.0539359336199197
1051    8       2       1       1.062602071647318
1061    10      2       0       1.0790378676276204
1067    5       4       2       1.0192233526522114
1083    3       5       3       1.022455144433677
1086    7       1       2       1.0744530079917134
1103    6       6       1       1.0444483384621503
1106    9       1       1       1.0924408760690854
1111    11      1       0       1.1034108596870509
1123    2       4       4       1.0287586375307691
1127    4       3       3       1.0184939554181438
1129    4       7       2       1.036728709204179
1132    7       4       1       1.0371421972174963
1133    8       6       0       1.077184161129713
1135    7       9       0       1.1304123868285583
1152    9       4       0       1.061759226951329
1157    3       2       4       1.0253228314541503
1219    3       6       3       1.0300503692625507
1222    6       3       2       1.0290517213518562
1228    5       5       2       1.021222894287424
1239    6       7       1       1.0520101536093853
1241    7       10      0       1.1458333962853755
1269    2       5       4       1.034750129380506
1272    5       2       3       1.0340739931281993
1272    8       3       1       1.0472719155273764
1274    8       7       0       1.0876684205758178
1287    10      3       0       1.066365416363215
1292    4       1       4       1.0435845857216208
1308    4       4       3       1.0161643477833935
1308    7       5       1       1.0375335867295665
1333    9       5       0       1.0643406477882664
1342    7       2       2       1.050293981210886
1353    3       3       4       1.0197899160437187
1372    6       1       3       1.0610194354429587
1373    2       2       5       1.0270770779909042
1375    6       8       1       1.061882285301361
1377    9       2       1       1.0678744951246646
1387    11      2       0       1.0821334358822232
1389    5       6       2       1.0242098799174948
1415    8       8       0       1.0997022750365375
```

1417	8	1	2	1.0803752196495442
1428	6	4	2	1.0226736500888474
1437	10	1	1	1.09642583626826
1442	12	1	0	1.1057510153710501
1484	7	6	1	1.041448880720697
1489	4	5	3	1.0169152950052776
1493	8	4	1	1.0402550366636496
1503	5	3	3	1.0223834035124477
1511	6	9	1	1.0725885670785242
1513	10	4	0	1.0618607823241757
1514	9	6	0	1.0703570718395008
1543	4	2	4	1.028669742178452
1549	3	4	4	1.020458041226654
1550	5	7	2	1.0304918363333329
1556	8	9	0	1.1120844108255412
1579	2	3	5	1.0263273270953015
1598	7	3	2	1.0347988166488296
1634	6	5	2	1.0213832605991042
1648	9	3	1	1.0518847830477651
1658	6	2	3	1.04047671541216
1660	7	7	1	1.0478079233320257
1663	11	3	0	1.068509509264896
1670	4	6	3	1.0222810410352614
1683	5	1	4	1.0500156112223789
1695	9	7	0	1.079000418460763
1697	8	10	0	1.1251255760646024
1711	5	8	2	1.0386964487236938
1714	8	5	1	1.0388162005567747
1728	8	2	2	1.0565848800253803
1734	5	4	3	1.0170564488352196
1739	10	5	0	1.062074539629052
1745	3	5	4	1.0223103918919954
1763	7	1	3	1.0676802115713389
1763	10	2	1	1.072662271304186
1773	12	2	0	1.0850816834668164
1785	2	4	5	1.0293106445789004
1794	4	3	4	1.0208388422462893
1808	9	1	2	1.0855289629987501
1824	3	2	5	1.0272670118051637
1828	11	1	1	1.0997082954341528
1833	13	1	0	1.1079399127435683
1836	7	8	1	1.0559859895289785
1838	8	11	0	1.1384910393672527
1840	6	6	2	1.0234579305939115
1851	4	7	3	1.0276521150675095
1854	7	4	2	1.0272755406664253
1876	9	8	0	1.0882459728176859
1919	9	4	1	1.043854561152064
1935	8	6	1	1.0409517136354511
1939	11	4	0	1.062477092244881
1941	3	6	4	1.0272831228486934
1944	6	3	3	1.027370359343128
1965	5	5	3	1.0169395149817424
1965	10	6	0	1.0661316854686624

```
1991    2       5       5       1.0341224979777712
1994    5       2       4       1.0337913715184734
```

Second, a number of `XX_Max_Size_YYYY_Clusters_interpolated_at_size_ZZZ.txt` are generated, where `ZZZ` are the size of the cluster that you wanted to obtain information about how to create. These are the number of atoms that are given in `output_information['Sizes to obtain instructions to create clusters for']` in the `Interpolation_Script.py` script (see Output information for the interpolation scheme). These text files give the information required to create all the various symmetric and unsymmetric icosahedral, decahedral, and octahedral clusters that may have comparable energies as predicted by NISP. An example for how to make Au nanoclusters with 742 atoms, using the RGL potetial with parameters from Baletto *et al.* (DOI: 10.1063/1.1448484[38]), is shown below:

```
-------------------------------------
-------------------------------------
Icosahedral Interpolation
motif: Icosahedron, type_of_connection: ico, Remove atoms from: 111 Facet, motif␣
↪start details: [7], motif end details: [6], interpolation energy: 1.
↪1212719334692314.
Number of atoms to remove from ico [7]: 181
-------------------------------------
-------------------------------------
Decahedral Interpolation
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [4, 5, 2], motif end details: [4, 3, 2], interpolation energy: 1.
↪0203822945780565.
Number of atoms to remove from deca [4, 5, 2]: 145
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [3, 4, 3], motif end details: [3, 2, 3], interpolation energy: 1.
↪0209016834911688.
Number of atoms to remove from deca [3, 4, 3]: 205
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [4, 4, 2], motif end details: [4, 2, 2], interpolation energy: 1.
↪0230249158509677.
Number of atoms to remove from deca [4, 4, 2]: 24
motif: Decahedron, type_of_connection: reent, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [4, 4, 2], motif end details: [2, 4, 3],␣
↪interpolation energy: 1.023273967913156.
Number of atoms to remove from deca [4, 4, 2]: 24
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [4, 4, 2], motif end details: [5, 5, 1],␣
↪interpolation energy: 1.028265058828974.
Number of atoms to remove from deca [4, 4, 2]: 24
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [3, 3, 3], motif end details: [3, 1, 3], interpolation energy: 1.
↪0282797597217193.
Number of atoms to remove from deca [3, 3, 3]: 69
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [2, 3, 4], motif end details: [2, 1, 4], interpolation energy: 1.
↪0284674139453696.
Number of atoms to remove from deca [2, 3, 4]: 235
motif: Decahedron, type_of_connection: reent, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [5, 2, 2], motif end details: [3, 2, 3],␣
↪interpolation energy: 1.0295345996123988.
Number of atoms to remove from deca [5, 2, 2]: 3
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [5, 2, 2], motif end details: [6, 3, 1],␣
↪interpolation energy: 1.0366803032223728.
```

(continues on next page)

[38] https://doi.org/10.1063/1.1448484

(continued from previous page)

```
Number of atoms to remove from deca [5, 2, 2]: 3
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [6, 5, 1], motif end details: [6, 3, 1], interpolation energy: 1.
↪037355048803492.
Number of atoms to remove from deca [6, 5, 1]: 225
motif: Decahedron, type_of_connection: reent, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [4, 1, 3], motif end details: [2, 1, 4],␣
↪interpolation energy: 1.0391760455537304.
Number of atoms to remove from deca [4, 1, 3]: 23
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [5, 6, 1], motif end details: [5, 4, 1], interpolation energy: 1.
↪0407789908324858.
Number of atoms to remove from deca [5, 6, 1]: 45
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [5, 3, 2], motif end details: [5, 1, 2], interpolation energy: 1.
↪0418464244263257.
Number of atoms to remove from deca [5, 3, 2]: 164
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [6, 4, 1], motif end details: [6, 2, 1], interpolation energy: 1.
↪0423694742674026.
Number of atoms to remove from deca [6, 4, 1]: 89
motif: Decahedron, type_of_connection: reent, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [5, 6, 1], motif end details: [3, 6, 2],␣
↪interpolation energy: 1.04313742776245.
Number of atoms to remove from deca [5, 6, 1]: 45
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [5, 7, 1], motif end details: [5, 5, 1], interpolation energy: 1.
↪0490249094114161.
Number of atoms to remove from deca [5, 7, 1]: 146
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [6, 4, 1], motif end details: [7, 5, 0],␣
↪interpolation energy: 1.0550989365941636.
Number of atoms to remove from deca [6, 4, 1]: 89
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [7, 2, 1], motif end details: [8, 3, 0],␣
↪interpolation energy: 1.0602793413245615.
Number of atoms to remove from deca [7, 2, 1]: 38
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [7, 3, 1], motif end details: [7, 1, 1], interpolation energy: 1.
↪0626955603486183.
Number of atoms to remove from deca [7, 3, 1]: 214
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [8, 5, 0], motif end details: [8, 3, 0], interpolation energy: 1.
↪0661515623770428.
Number of atoms to remove from deca [8, 5, 0]: 250
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [8, 4, 0], motif end details: [8, 2, 0], interpolation energy: 1.
↪0682841146341562.
Number of atoms to remove from deca [8, 4, 0]: 109
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [8, 4, 0], motif end details: [7, 5, 0],␣
↪interpolation energy: 1.0695080161043045.
Number of atoms to remove from deca [8, 4, 0]: 109
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [9, 2, 0], motif end details: [8, 3, 0],␣
↪interpolation energy: 1.069856934349611.
Number of atoms to remove from deca [9, 2, 0]: 48
```

(continues on next page)

(continued from previous page)

```
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [7, 6, 0], motif end details: [7, 4, 0], interpolation energy: 1.
↪0772609639389423.
Number of atoms to remove from deca [7, 6, 0]: 75
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [9, 3, 0], motif end details: [9, 1, 0], interpolation energy: 1.
↪0814963971429128.
Number of atoms to remove from deca [9, 3, 0]: 229
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [5, 6, 1], motif end details: [6, 7, 0],␣
↪interpolation energy: 1.084236258131725.
Number of atoms to remove from deca [5, 6, 1]: 45
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [7, 7, 0], motif end details: [7, 5, 0], interpolation energy: 1.
↪0882615676069058.
Number of atoms to remove from deca [7, 7, 0]: 181
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [5, 7, 1], motif end details: [6, 8, 0],␣
↪interpolation energy: 1.0985691607535673.
Number of atoms to remove from deca [5, 7, 1]: 146
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [7, 6, 0], motif end details: [6, 7, 0],␣
↪interpolation energy: 1.1037261745238776.
Number of atoms to remove from deca [7, 6, 0]: 75
motif: Decahedron, type_of_connection: plane, Remove atoms from: 100 Facet or from␣
↪Corners, motif start details: [7, 7, 0], motif end details: [6, 8, 0],␣
↪interpolation energy: 1.1194332919972068.
Number of atoms to remove from deca [7, 7, 0]: 181
motif: Decahedron, type_of_connection: deca_111, Remove atoms from: 111 Facet, motif␣
↪start details: [6, 9, 0], motif end details: [6, 7, 0], interpolation energy: 1.
↪1381777464650709.
Number of atoms to remove from deca [6, 9, 0]: 47
------------------
The deca motif has a cluster(s) with the exact number of atoms.
No. Atoms: 742.
Motif Type: Decahedron.
Motif details: [2, 5, 3].
-------------------------------------
-------------------------------------
Octahedral Interpolation
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [12, 3], motif end details: [10, 1], interpolation energy: 1.
↪022091520879608.
Number of atoms to remove from octa [12, 3]: 330
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [12, 4], motif end details: [10, 2], interpolation energy: 1.
↪022186081865912.
Number of atoms to remove from octa [12, 4]: 234
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [11, 3], motif end details: [9, 1], interpolation energy: 1.
↪0223228243474467.
Number of atoms to remove from octa [11, 3]: 65
motif: Octahedron, type_of_connection: octa_fcc, Remove atoms from: 100 Facet, motif␣
↪start details: [11, 3], motif end details: [11, 4], interpolation energy: 1.
↪0277370571331692.
Number of atoms to remove from octa [11, 3]: 65
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [11, 2], motif end details: [9, 0], interpolation energy: 1.
↪0296690286032208.
```

(continues on next page)

(continued from previous page)

```
Number of atoms to remove from octa [11, 2]: 119
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [12, 2], motif end details: [10, 0], interpolation energy: 1.
↪029920955385351.
Number of atoms to remove from octa [12, 2]: 384
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [11, 1], motif end details: [9, 0], interpolation energy: 1.
↪0343375753840016.
Number of atoms to remove from octa [11, 1]: 143
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [12, 1], motif end details: [10, 0], interpolation energy: 1.
↪0344036720798044.
Number of atoms to remove from octa [12, 1]: 408
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [11, 0], motif end details: [10, 0], interpolation energy: 1.
↪038707375707145.
Number of atoms to remove from octa [11, 0]: 149
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [12, 5], motif end details: [10, 3], interpolation energy: 1.
↪0438947041036042.
Number of atoms to remove from octa [12, 5]: 84
motif: Octahedron, type_of_connection: octa_111, Remove atoms from: 111 Facet, motif␣
↪start details: [13, 6], motif end details: [11, 4], interpolation energy: 1.
↪0727769381223602.
Number of atoms to remove from octa [13, 6]: 181
-------------------------------------
-------------------------------------
```

6.11 Helpful Programs to run NISP

Included in NISP are programs to help to execute all the VASP calculations that you want to run on slurm if you are using VASP to obtain all the energies of your clusters. 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_submitSL_slurm.py` - How to execute all Trials using the JobArray Slurm Job Submission Scheme*

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

If you installed NISP through pip3

If you installed the NISP 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_NISP="<Path_to_NISP>"
```

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

```
export PATH="$PATH_TO_NISP"/NISP/Subsidiary_Programs:$PATH
```

See more about this in *Installation of NISP*.

6.11.2 `Run_submitSL_slurm.py` - How to execute all Trials using the JobArray Slurm Job Submission Scheme

It is possible to perform this interpolation scheme using the energies of nanoclusters that have be obtained with DFT using VASP. If you select your `'Local Optimiser'` input in your `input_information` dictionary to be `'VASP'`, NISP will create a bunch of icosahedral, decahedral, and octahedral nanoclusters that are setup to run VASP calculations upon. This includes a POSCAR and submit.sl file, as well as POTCAR, INCAR, and KPOINTS files that you provide. Many nanoclusters are created to be used in the interpolation scheme, and therefore a lot of submit.sl file to submit to slurm.

To make this easier for the user, NISP has been included with the `Run_submitSL_slurm.py` script, which is designed to submit all your VASP calculations to slurm. This script will walk through all subdirectories from the directory you run this from and will run all your submit.sl scripts for you automatically. All you need to do is to move into the `VASP_Clusters` folder that has been created by slurm and enter `Run_submitSL_slurm.py` into the terminal, press enter, and watch all your NISP nanocluster be submitted to slurm.

There is one input that you can pass into the program. The `Run_mass_submitSL_slurm.py` program is designed to wait one minute between submitting jobs. This is because you can cause slurm issues if too many jobs are submitted at once. However, you can override this if you enter in `False` as the first argument. This will tell `Run_submitSL_slurm.py` to keep submitting jobs without waiting one minute between them.

Note that you will notice that if you submit too many jobs, or if you submit too many jobs too quickly, you will get problems with slurm that you are submitting too many jobs or submitting them too quickly. This is all dependent on how slurm is set up. This has been covered by the program, and will tell you how long it will be waiting when it is waiting. Do not stop the program while it is submitting jobs. If you do quit before it has finished, it will make resubmitting jobs difficult.

The following variables can be different for different people. Feel free to change the following variables in your `Subsidiary_Programs/Run_submitSL_slurm.py` as you need

- **Max_jobs_in_queue_at_any_one_time** (*int*): This is the maximum number of jobs that slurm allows you to have in your queue. This is usually set by default to 1000. I personally have set this to 10,000 and this is what is current set in `Run_submitSL_slurm.py`. Default: 10,000
- **time_to_wait_before_next_submission** (*float*): This is the amount of time that this program waits after submitting a job, before continuing on. This is given in seconds. **Do not set this to less than 10 seconds.** Default: 20.0 (seconds)
- **time_to_wait_max_queue** (*float*): This is the amount of time that this program waits after it has found that the maximum number of jobs have been submitted to the queue. `Run_submitSL_slurm.py` will wait for this

amount of time before continuing again. This is given in seconds. **Do not set this to less than 10 seconds.** Default: 60.0 (seconds)

Problems can occur every so often when submitting jobs to slurm, but these are generally internet connectivity problems or slurm hanging problems that resolve themselves after a few tens of seconds. There are two other variables that determine how `Run_submitSL_slurm.py` will deal with issues.

- **time_to_wait_before_next_submission_due_to_temp_submission_issue** (*float*): This is the amount of time that this program waits after it has experienced an error in submitting a job. This is given in seconds. **Do not set this to less than 10 seconds.** Default: 10.0 (seconds)
- **number_of_consecutive_error_before_exitting** (*int*): This is the number of times that `Run_mass_submitSL_slurm.py` will attempt to resubmit a job to slurm before it will give up. After this many consecutive errors arising, some systematic error is likely occuring. In this case, `Run_mass_submitSL_slurm.py` will print the directories of all the jobs that were not submitted and then close.

Hopefully running `Run_submitSL_slurm.py` will submit all your genetic algorithm trials.

The names of the jobs can be quite big. When looking in `squeue` to see how things are going, it is sometimes useful to expand the names in the squeue output. This can be done as shown below:

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

Here, after `-o`, `i` specifies the job ID and `j` specifies the job name. You can change this number to the number of characters this will display. Here `%.20i` indicates that `squeue` will dedicate 20 characters to displaying the job ID and `%.50j` indicates that `squeue` will dedicate 50 characters to displaying the name of the job.

6.12 NISP Python Files

Below are all the files that are used by NISP to run.

6.12.1 Table of Contents

Run_Interpolation_Scheme.py

This method runs the NISP program.

This program is designed to run and give the results of the interpolation scheme as described in A. L. Garden, A. Pedersen, H. Jónsson, "Reassignment of 'magic numbers' of decahdral and FCC structural motifs", Nanoscale, 10, 5124-5132 (2018).

Cluster.py

This class is designed to hold all the information we need to record about clusters: Inputs (str) motif - the motif you want to get the number of atoms required to create it. (list) motif_details - the information required for the motif of interest (atoms) cluster - A cluster to sample (int) no_atoms - the number of atoms in the cluster (Calculator) calculator - The calculator to use (float) e_coh - The cohesive energy of the cluster (float) delta_energy - The delta energy of the cluster. (boolean) debug - Print any information to help us if we need to. Output (int) noAtoms - the number of atoms required to make the cluster of interest

motif_methods.py

This script contains definition about motifs, such as the number of atomd a cluster contains with certain specifications.

Interpolation_Connection.py

This contains a class that writes the information about a connection between two clusters of the same motif together.

Interpolation_rules.py

This method contains all the definitions about the rules used to connect clusters of a particular motif together.

Counter.py

This class acts like a Integer and allows one to add to the counter.

6.13 Index

6.14 Python Module Index

CHAPTER

SEVEN

INDICES AND TABLES

- *Index*
- *Python Module Index*

PYTHON MODULE INDEX

n

INDEX

M

N