The Adsorber Documentation

Release 1

Geoffrey Weal

Dec 30, 2021

CONTENTS

1 2 3 4 5 6

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/Adsorber
[3] https://pypi.org/project/Adsorber/
[4] https://anaconda.org/GardenGroupUO/Adsorber
[5] https://www.gnu.org/licenses/agpl-3.0.en.html
[6] https://lgtm.com/projects/g/GardenGroupUO/Adsorber/context:python

CHAPTER

ONE

WHAT IS ADSORBER

Adsorber is a program designed to adsorb molecules in various way to the surface of a cluster or a surface. This program adsorbed molecules ontop of atoms, ontop of bridge sites, ontop of three-fold sites, and ontop of four fold sites.

A guide on how to use the `Adsorber` program is given in *Guide To Using Adsorber*.

This program is definitely a "work in progress". I have made it as easy to use as possible, but there are always oversights to program development and some parts of it may not be as easy to use as it could be. If you have any issues with the program or you think there would be better/easier ways to use and implement things in `Adsorber`, feel free to email Geoffrey about these (geoffrey.weal@gmail.com). Feedback is very much welcome!

CHAPTER

TWO

INSTALLATION

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

As well as installing Adsorber, the Atomic Simulation Environment (ASE) GUI and Jmol programs are also used to visualise your system with adsorbed atoms and molecules upon it. Installation and how to use the ASE GUI and Jmol can be found in *External Programs that will be useful to install for using Adsorber*.

CHAPTER THREE

GUIDE TO USING ADSORBER

After you have installed Adsorber and all other helpful programs, see *Guide To Using Adsorber* to learn about the process of using Adsorber to obtain models of your system with various adsorbates for further optimisation with VASP.

CHAPTER FOUR

TABLE OF CONTENTS

4.1 Installation: Setting Up Adsorber and Pre-Requisites Packages

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

4.1.1 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
------------------------------ ---------
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
```

(continues on next page)

(continued from previous page)

```
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[7]

Atomic Simulation Environment

Adsorber uses the atomic simulation environment (ASE) to create models of clusters and surfaces that have atoms and moleucles adsorbed to its surface. Read more about ASE here[8].

The installation of ASE can be found on the ASE installation page[9], 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
```

[7] https://pip.pypa.io/en/stable/installing/
[8] https://wiki.fysik.dtu.dk/ase/
[9] https://wiki.fysik.dtu.dk/ase/install.html

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

Networkx

`Networkx` is a python program that is used in `Adsorber` to determine if two systems are structurally identical. The easiest way to install `Networkx` is though pip. Type the following into the terminal:

```
pip3 install --upgrade --user networkx
```

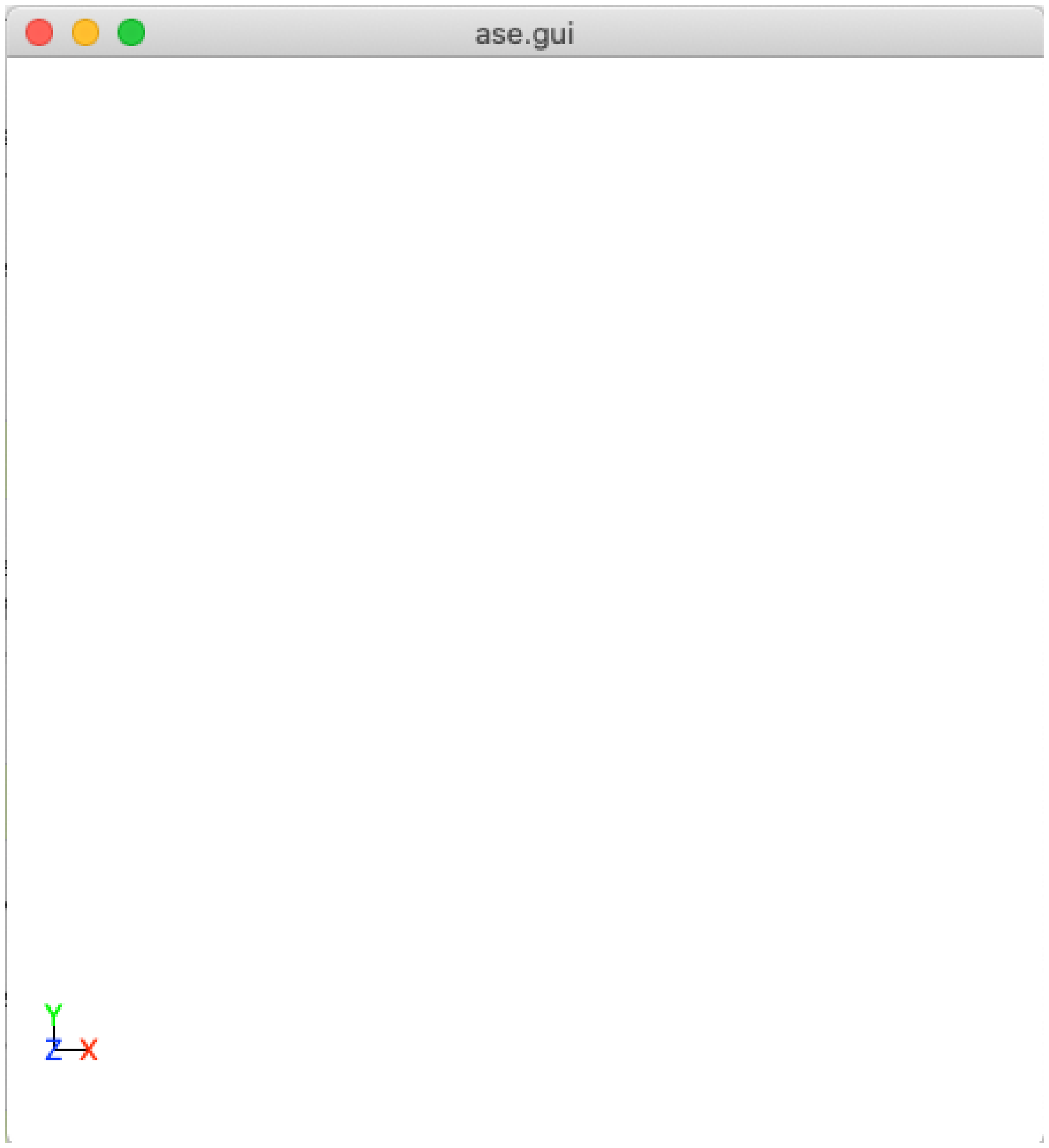


Fig. 1: This is a blank ase gui screen that you would see if enter `ase gui` into the terminal.

Openpyxl

`Openpyxl` is a python program that is used in `Adsorber` to write excel spreadsheets of information of your clusters and surface models with adsorbates attach once they have been locally optimised with VASP. The easiest way to install `Openpyxl` is though pip. Type the following into the terminal:

```
pip3 install --upgrade --user openpyxl
```

Packaging

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

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

4.1.2 Setting up Adsorber

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

Install Adsorber through `pip3`

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

```
pip3 install --upgrade --user Adsorber
```

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

[10]

Install Adsorber through `conda`

You can also install Adsorber through `conda`, however I am not as versed on this as using `pip3`. See docs.conda.io[11] to see more information about this. Once you have installed anaconda on your computer, I believe you install Adsorber using `conda` by performing the following in your terminal.

```
conda install ase
conda install adsorber
```

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

[12]

[10] https://pypi.org/project/Adsorber/
[11] https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html
[12] https://anaconda.org/GardenGroupUO/adsorber

Manual installation

First, download Adsorber 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_Adsorber_TO_LIVE_ON_YOUR_COMPUTER
git clone https://github.com/GardenGroupUO/Adsorber
```

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 Adsorber program into `pwd`

```
pwd
```

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

```
export PATH_TO_Adsorber="<Path_to_Adsorber>"
export PYTHONPATH="$PATH_TO_Adsorber":$PYTHONPATH
```

where `"<Path_to_Adsorber>"` is the directory path that you place Adsorber (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 Adsorber, the `Run_Adsorber.py` file.

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

```
export PATH="$PATH_TO_Adsorber"/Adsorber/Subsidiary_Programs:$PATH
```

4.1.3 Visualisation Programs for looking at systems with adsorbed molecules

As well as installing Adsorber, the Atomic Simulation Environment (ASE) GUI and Jmol programs are also used to visualise your system with adsorbed atoms and molecules upon it. Installation and how to use the ASE GUI and Jmol can be found in *External Programs that will be useful to install for using Adsorber*.

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
```

There are also two aliases that are useful, these are

- `no_of_jobs_running_or_queued`: Will indicate the number of jobs that are either running or in the queue in slurm.
- `no_of_submitSL_files`: Will give the number of VASP models in subdirectories that are to be run. These should all contain submit.sl files, which is what this alias is doing.

These two aliases are given below for you to also add to your `~/.bashrc`:

```
alias no_of_jobs_running_or_queued="squeue -u $USER | wc -l"
alias no_of_submitSL_files='find . -name "submit.sl" -type f -not -path "*Submission_
↪Folder_*" | wc -l'
```

These two aliases are explained further in *How to submit VASP jobs to Slurm: The Run_Adsorber_submitSL_slurm.py program*.

Note that the `no_of_jobs_running_or_queued` reference will give you the number of jobs you have submitted to slurm, plus 1. So whatever number you get from `no_of_jobs_running_or_queued`, minus 1 from it to get the number of jobs in your slurm queue. Don't know how to fix this yet.

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 Adsorber

export PATH_TO_Adsorber="<Path_to_Adsorber>"
export PYTHONPATH="$PATH_TO_Adsorber":$PYTHONPATH
export PATH="$PATH_TO_Adsorber"/Adsorber/Subsidiary_Programs:$PATH

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

alias no_of_jobs_running_or_queued="squeue -u $USER | wc -l"
alias no_of_submitSL_files='find . -name "submit.sl" -type f -not -path "*Submission_
↪Folder_*" | wc -l'

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

4.2 External Programs that will be useful to install for using `Adsorber`

There will be two programs that we will use to view clusters/surface models that we will use and create in `Adsorber`.

4.2.1 ASE GUI (from Atomic Simulation Environment)

The ASE GUI is a program that is use to view clusters in ASE. It is build very simply and is perfect for looking at all of the models we will be making and using for `Adsorber`.

The ASE GUI runs through ASE. To install ASE, see *Atomic Simulation Environment*. Once install, you can type the following into the terminal to show the ASE GUI:

```
ase gui
```

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

You can then look at your surface model by entering in the following into the terminal, where `name_of_your_file` is the name of the file of your cluster/surface model that you wish to look at in ASE GUI.

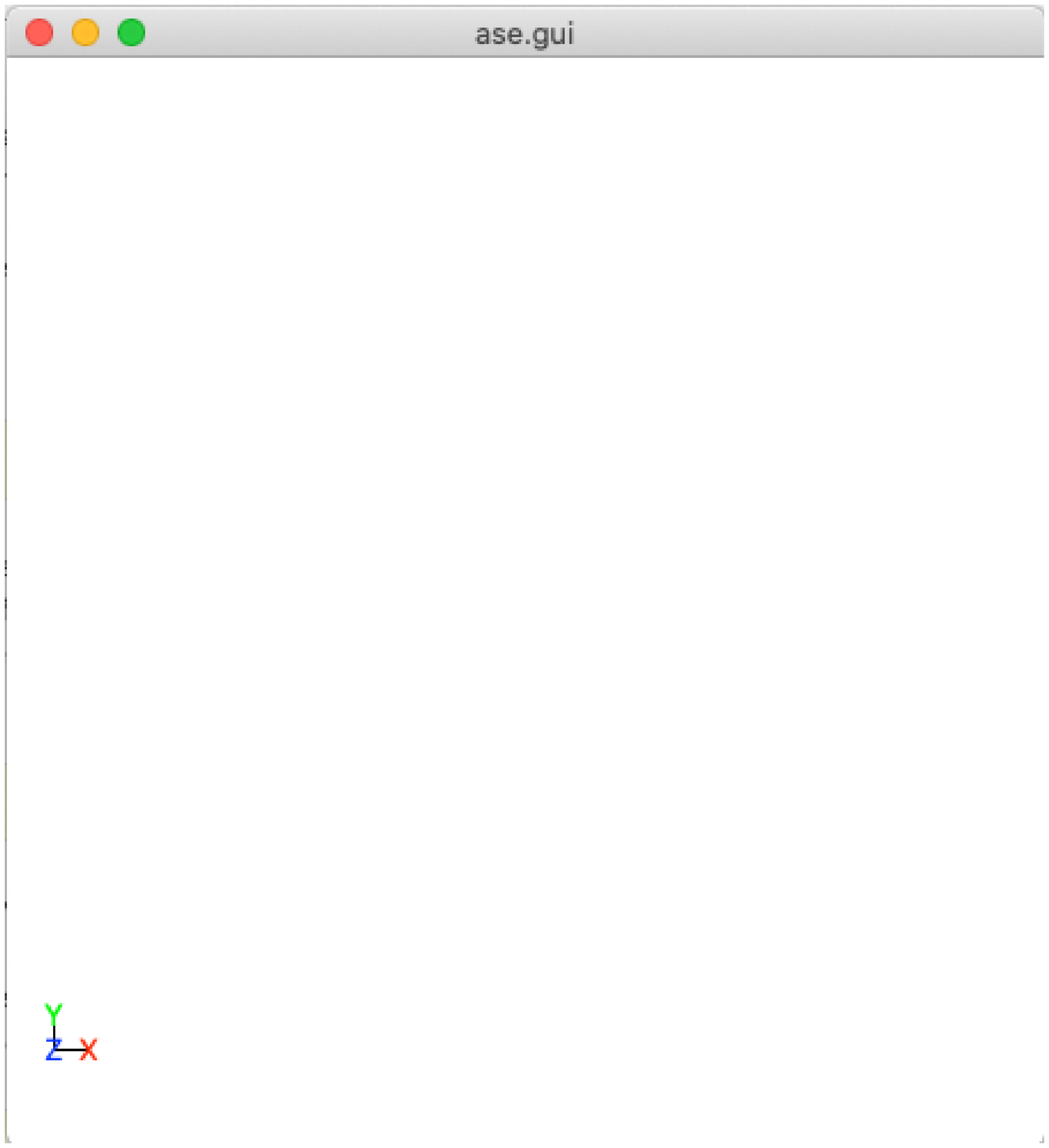


Fig. 2: This is a blank ase gui screen that you would see if enter `ase gui` into the terminal.

```
ase gui name_of_your_file
```

You will then see your model in ASE GUI. An example of this is shown below:

4.2.2 Jmol

Jmol is another program that is used to visualise chemical systems. See Jmol Web Page[13] for information about how to download Jmol on your computer. Once you have downloaded Jmol, add this to your `~/.bashrc` for each access to Jmol in your terminal

```
function jmol(){ /Path_to_Jmol/jmol-VERSION_NO/jmol.sh $* >/dev/null & }
```

where `Path_to_Jmol` is the path to your Jmol folder that you have just downloaded and untarred/unzipped and `jmol-VERSION_NO` is the folder that Jmol is in (that you have just downloaded and untarred/unzipped).

To test that Jmol is working, type into the terminal:

```
jmol
```

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

You can then look at your surface model by entering in the following into the terminal, where `name_of_your_file` is the name of the file of your cluster/surface model that you wish to look at in Jmol.

```
jmol name_of_your_file
```

You will then see your model in Jmol GUI. An example of this is shown below:

4.3 Guide To Using Adsorber

There are four parts (Part A, Part B, Part C + Part D) + a prelude step to using the Adsorber program. These parts are described in the sections below:

4.3.1 Prelude Step

In this step, we will step up the `Run_Adsorber.py` script, which includes selecting which atoms in your system are surface atoms. The information about how to set up the `Run_Adsorber.py` script is given in *Prelude 1: Run_Adsorber.py - Running Adsorber*. Instructions of some of the more involved settings that are required in the `Run_Adsorber.py` script are given in *Prelude 2: How to obtain some of the settings for the Run_Adsorber.py script*. These include:

- indicating which atoms are surface atoms, as Adsorber does not identify surface atoms. This is required for Adsorber to know which atoms to consider when placing adsorbates onto top sites, bridging sites, three-fold sites, and four-fold sites. You can find out how to specify this in *How to Mark Surface Atoms in your Cluster/Surface model in Adsorber*.
- indicating how adsorbates are adsorbed to your system. This is given in *How to Bind Molecule to the Surface of your Cluster/Surface Model in Adsorber*.

[13] http://jmol.sourceforge.net/

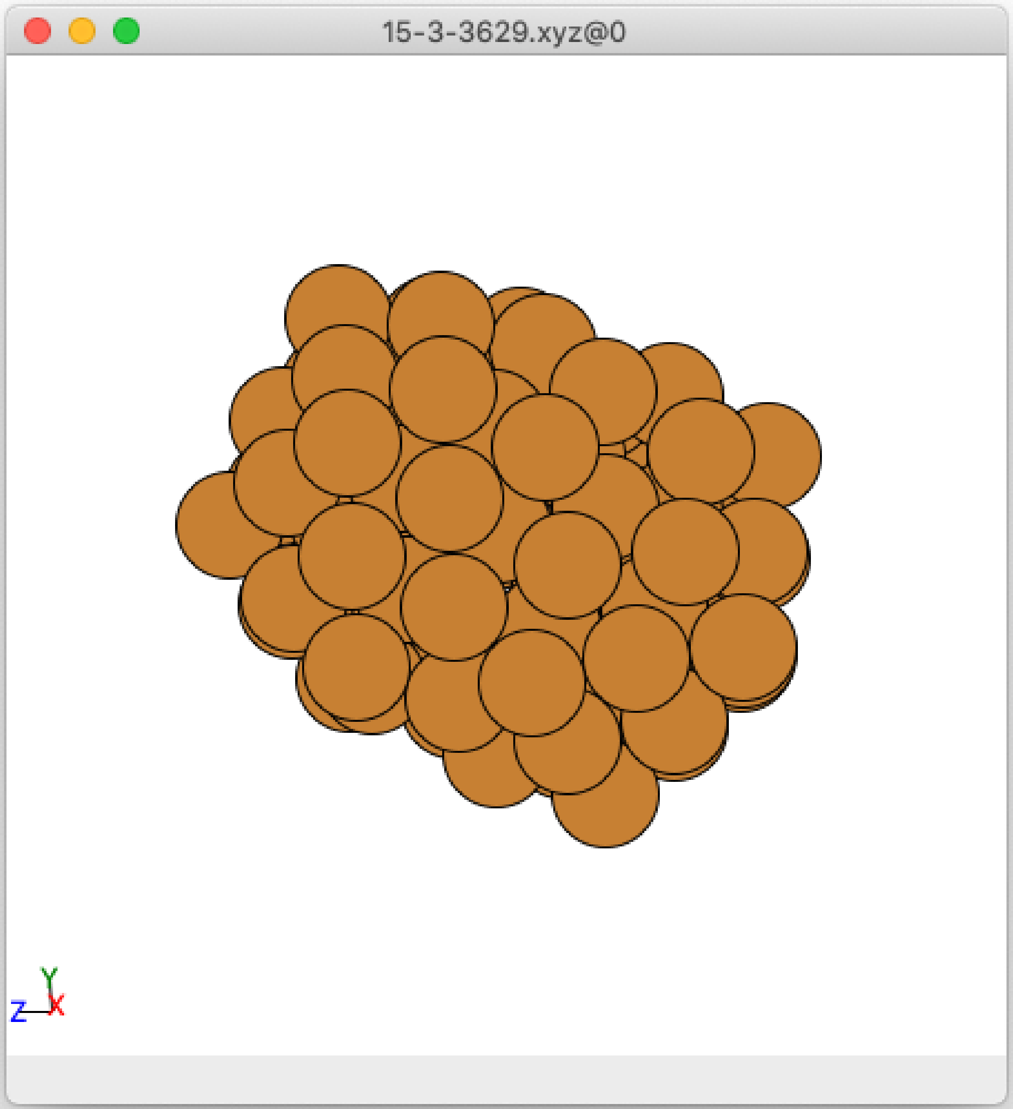


Fig. 3: This is the ase gui screen that shows an example of a cluster on it.

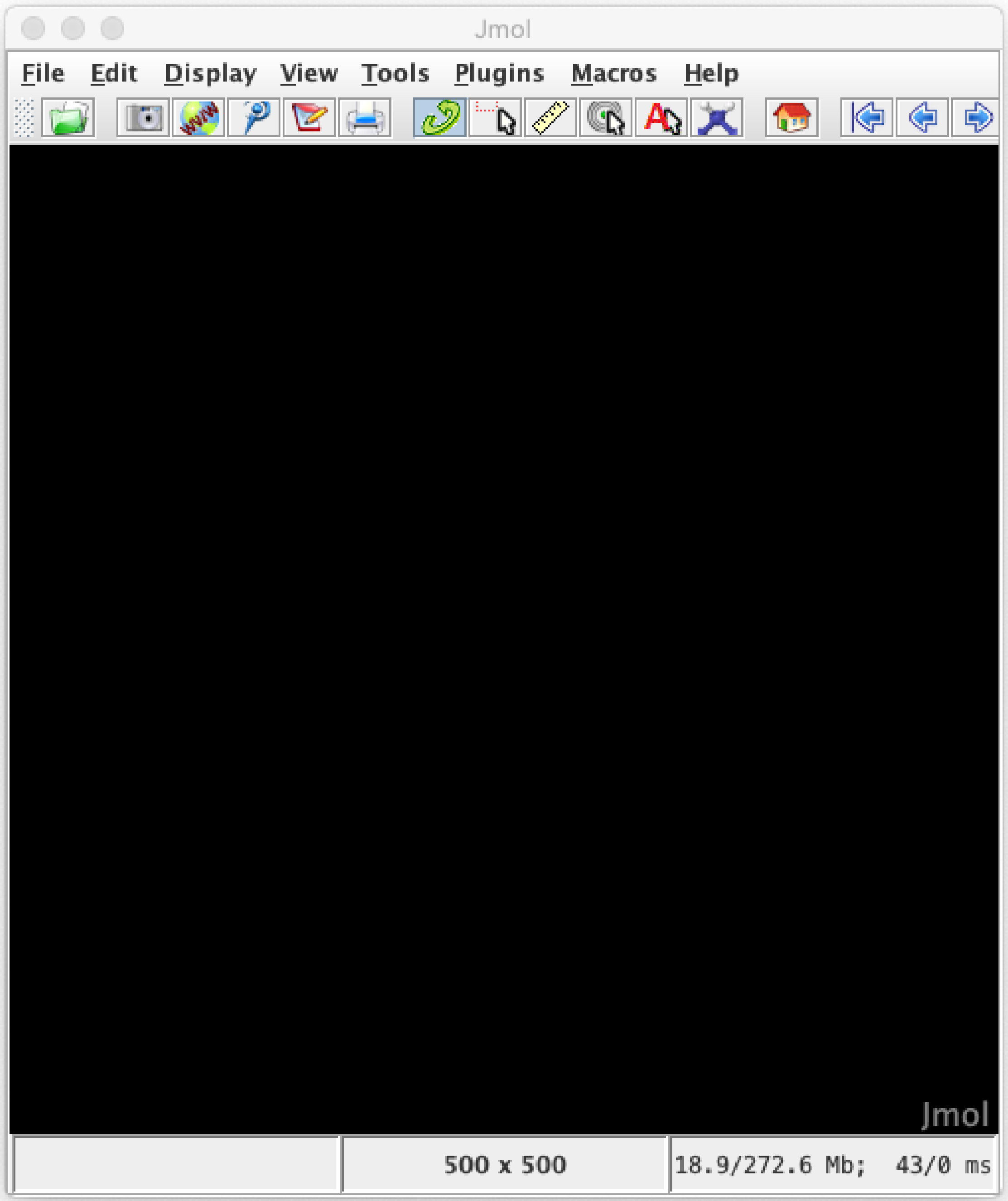


Fig. 4: This is a blank Jmol gui screen that you would see if enter `Jmol` into the terminal.

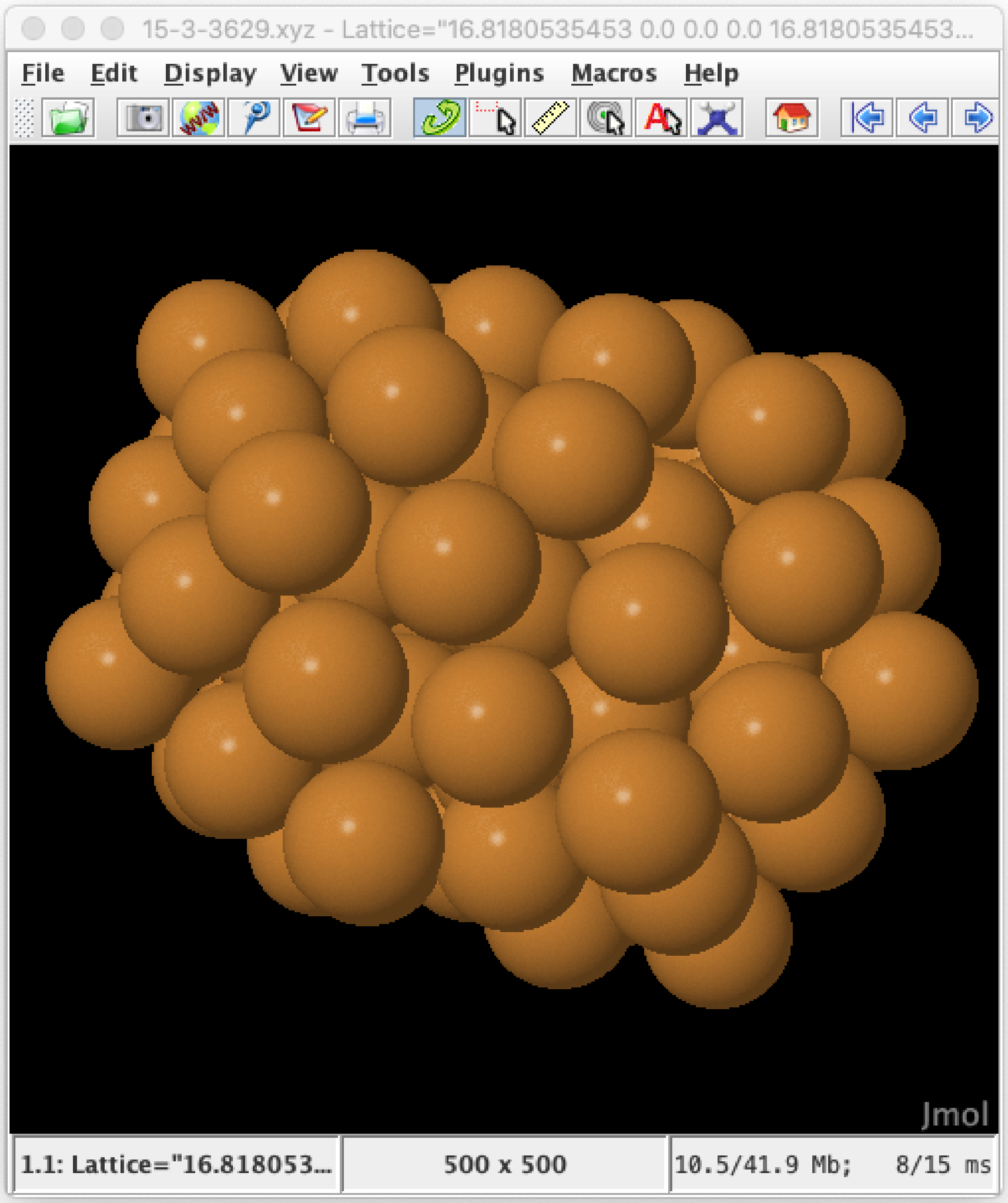

15-3-3629.xyz - Lattice="16.8180535453 0.0 0.0 0.0 16.8180535453...
File
Edit
Display
View
Tools
Plugins
Macros
Help
Jmol
1.1: Lattice="16.818053...
500 x 500
10.5/41.9 Mb; 8/15 ms

Fig. 5: This is the Jmol screen that shows an example of a cluster on it.

4.3.2 Part A

This part creates the VASP files to locally optimise your system. This is done first because this locally optimised system is used to adsorb adsorbate onto in Parts B and C. This part will also create the VASP files for locally optimising adsorbates (not bound to your system). These files are not used further in the Adsorber program, but you often need to know what the energy of you adsorbates are under the functional you are using, so this allows you to easily obtain the energies of these adsorbates. The guide to perform this set is given in *Part A: How to optimise your system initially*. If you already have the system locally optimised with your functional, feel free to move on to Part B.

4.3.3 Part B

In this part, your VASP optimised system will have all adsorbates attached to it in all the various top sites, bridging sites, three fold sites, and four fold sites found across the surface of your system. From these you can select which models your want to include for further optimisation by VASP. The guide to perform this part is given in *Part B: Adsorbing Adsorbates to your System*.

4.3.4 Part C

In this part, Adsorber will take the models of adsorbates on various sites on your system and create the files required to locally optimise these models in VASP. The guide to perform this part is given in *Part C.1: Preparing Selected Adsorbed Systems For VASP Optimisation*. Subsidiary programs have also been designed to resubmit jobs that have not completed or need to be converged tigher or looser. These are designed in *Part C.2: What To Do If Some Jobs Have Not Finished/Converged*.

4.3.5 Part D

In this part, a subsidiary program is used to gather information about your VASP local optimisations, including the energy of absorbate+system as well as adsorbates and system alone, as well as if the VASP local optimisation converged or not. The guide to perform this part is given in *Part D: Gathering Information from VASP Calculations*.

4.3.6 Final Notes about this process

You may want to move backwards and forwards between these parts as you progress in your study. Parts B - D can be performed multiple times if you want to include extra adsorbates in your study. I performed Part C multiple times as I like to focus on obtaining and understanding the energetics of an adsorbates on various top, bridging, three-fold, and four-fold sites and understand which of these site are preferable for binding before moving on to the next adsorbate.

4.4 Prelude 1: *Run_Adsorber.py* - Running Adsorber

In this article, we will look at how to run the `Adsorber` program. This program is run though the **Run_Adsorber.py** script, which includes all the information required for `Adsorber` to adsorb species to surface sites of surfaces and clusters. You can find an example of `Run_Adsorber.py` files at github.com/GardenGroupUO/Adsorber[14] under `Examples`.

[14] https://github.com/GardenGroupUO/Adsorber

4.4.1 Running the Adsorber Program

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

Listing 1: Run_Adsorber.py

```python
from ase.build import molecule
from Adsorber import Adsorber_Program

# ------------------------------------------------------------------------------------------------------------------------------------
# Initial inputs for the Adsorber program
part_to_perform = 'Part B'

# General Variables
cluster_or_surface_model = 'cluster'

# Part A information
system_filename = '15-3-3629_with_vacuum_6.0_Ang.xyz'

# Part B information
path_to_VASP_optimised_non_adsorbate_system = 'Part_A_Non_Adsorbed_Files_For_VASP/system/OUTCAR'
cutoff = 3.5
surface_atoms = [11,25,28,13,3,8,6,23,22,59,34,62,66,1,0,4,30,15,14,16,5,12,29,2,7,10,24,26,70,35,47,50,60,63,48,39,41,44,54,68,76,71,32,31,74,42,56,52,43,40,46,61,53,45,57,72,73,77]

# ------------------------------------------------------------------------------------------------------------------------------------
# Give the Atoms objects for the atoms and molecules you want to adsorb to your cluster or surface model
adsorbed_species = []

COOH_symmetric = molecule('HCOOH') # note the carbon is index 1
COOH_symmetric.center(vacuum=10.0)
del COOH_symmetric[4] # remove the hydrogen atom
COOH_symmetric_axis = (0.1,-1,0)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
COOH_symmetric_adsorbed_species = {'name': 'COOH_symmetric', 'molecule': COOH_symmetric, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 1, 'axis': COOH_symmetric_axis, 'rotations': rotations, 'sites_to_bind_adsorbate_to': ['Top_Sites','Bridge_Sites','Three_Fold_Sites']}
adsorbed_species.append(COOH_symmetric_adsorbed_species)
```

(continues on next page)

(continued from previous page)

```python

COOH_O_tilted = molecule('HCOOH') # note the carbon is index 1
COOH_O_tilted.center(vacuum=10.0)
del COOH_O_tilted[4] # remove the hydrogen atom
COOH_O_tilted_axis = (-0.4,-1,0)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
COOH_O_tilted_adsorbed_species = {'name': 'COOH_O_tilted', 'molecule': COOH_O_tilted,
↪'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 1,
↪'axis': COOH_O_tilted_axis, 'rotations': rotations, 'sites_to_bind_adsorbate_to':
↪'Top_Sites'}
#adsorbed_species.append(COOH_O_tilted_adsorbed_species)

CO = molecule('CO') # note the carbon is index 1
CO.center(vacuum=10.0)
CO_axis = 'z'
distance_of_adatom_from_surface = 1.5
CO_adsorbed_species = {'name': 'CO', 'molecule': CO, 'distance_of_adatom_from_surface
↪': distance_of_adatom_from_surface, 'index': 1, 'axis': CO_axis}
#adsorbed_species.append(CO_adsorbed_species)

# -----------------------------------------------
# option1
COH = molecule('H2COH') # note the carbon is index 0
COH.center(vacuum=10.0)
del COH[4] # remove the hydrogen atom
del COH[3] # remove the hydrogen atom
COH_axis = '-x'
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
COH_adsorbed_species = {'name': 'COH', 'molecule': COH, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': COH_axis, 'rotations
↪': rotations}
#adsorbed_species.append(COH_adsorbed_species)

#option2
CHO = molecule('HCO') # note the carbon is index 0
CHO.center(vacuum=10.0)
CHO_axis = (-(3.0**0.5)/2.0,-1.0/2.0,0)
rotations = 'automatic' #range(0,360,10)
CHO_adsorbed_species = {'name': 'CHO', 'molecule': CHO, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CHO_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CHO_adsorbed_species)
# -----------------------------------------------
# option1
CH2O = molecule('H2CO') # note the carbon is index 1
CH2O.center(vacuum=10.0)
CH2O_axis = 'x'
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
CH2O_adsorbed_species = {'name': 'CH2O', 'molecule': CH2O, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 1, 'axis': CH2O_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CH2O_adsorbed_species)

# option1
```

(continues on next page)

(continued from previous page)

```
CHOH = molecule('H2COH') # note the carbon is index 0
CHOH.center(vacuum=10.0)
del CHOH[3]
CHOH_axis = (-1,-1,0)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
CHOH_adsorbed_species = {'name': 'CHOH', 'molecule': CHOH, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CHOH_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CHOH_adsorbed_species)
# ---------------------------------------------
# new option 1
CH3O = molecule('CH3O') # note the oxygen is index 1
CH3O.center(vacuum=10.0)
CH3O_axis = '-y'
distance_of_adatom_from_surface = 1.5
CH3O_adsorbed_species = {'name': 'CH3O', 'molecule': CH3O, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 1, 'axis': CH3O_axis}
#adsorbed_species.append(CH3O_adsorbed_species)

# new option 2
CH2OH = molecule('CH3OH') # carbon is index 0
CH2OH.center(vacuum=10.0)
del CH2OH[2] # remove the hydrogen atom
CH2OH_axis = (1,-1,0)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
CH2OH_adsorbed_species = {'name': 'CH2OH', 'molecule': CH2OH, 'distance_of_adatom_
↪from_surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CH2OH_axis,
↪'rotations': rotations}
#adsorbed_species.append(CH2OH_adsorbed_species)
# ---------------------------------------------
# new new option 1
CH2 = molecule('CH4') # carbon is index 0
CH2.center(vacuum=10.0)
del CH2[4] # remove the hydrogen atom
del CH2[3] # remove the hydrogen atom
CH2_axis = 'z'
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,180,10)
CH2_adsorbed_species = {'name': 'CH2', 'molecule': CH2, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CH2_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CH2_adsorbed_species)

CH3 = molecule('CH4') # carbon is index 0
CH3.center(vacuum=10.0)
del CH3[4] # remove the hydrogen atom
CH3_axis = (1,-1,1)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,120,10)
CH3_adsorbed_species = {'name': 'CH3', 'molecule': CH3, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CH3_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CH3_adsorbed_species)

# new new option 2
```

(continues on next page)

(continued from previous page)

```
O = molecule('O')
O.center(vacuum=10.0)
distance_of_adatom_from_surface = 1.5
O_adsorbed_species = {'name': 'O', 'molecule': O, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface}
#adsorbed_species.append(O_adsorbed_species)

OH = molecule('OH') # note the oxygen is index 0
OH.center(vacuum=10.0)
OH_axis = '-z'
distance_of_adatom_from_surface = 1.5
OH_adsorbed_species = {'name': 'OH', 'molecule': OH, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 0, 'axis': OH_axis}
#adsorbed_species.append(OH_adsorbed_species)

C = molecule('C')
C.center(vacuum=10.0)
distance_of_adatom_from_surface = 1.5
C_adsorbed_species = {'name': 'C', 'molecule': C, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface}
#adsorbed_species.append(C_adsorbed_species)

# ------------------------------------------------------------------------------------------------------------------------------
# Add here any other atoms and molecules that you need to locally optimise in VASP for energy calculations
Other_molecules_to_obtain_VASP_energies_for = []

graphene = molecule('C')
graphene.center(vacuum=10.0)
graphene_optimise_energy = {'name': 'graphene', 'molecule': graphene}
Other_molecules_to_obtain_VASP_energies_for.append(graphene_optimise_energy)

H2 = molecule('H2')
H2.center(vacuum=10.0)
H2_optimised_energy = {'name': 'H2', 'molecule': H2}
Other_molecules_to_obtain_VASP_energies_for.append(H2_optimised_energy)

H2O = molecule('H2O')
H2O.center(vacuum=10.0)
H2O_optimised_energy = {'name': 'H2O', 'molecule': H2O}
Other_molecules_to_obtain_VASP_energies_for.append(H2O_optimised_energy)

# ------------------------------------------------------------------------------------------------------------------------------
# slurm informaion for making the submit.sl files for submitting VASP jobs in slurm

slurm_information = {}
slurm_information['project'] = 'uoo02568'
slurm_information['partition'] = 'large'
slurm_information['time'] = '72:00:00'
slurm_information['nodes'] = 1
slurm_information['ntasks_per_node'] = 12
slurm_information['mem-per-cpu'] = '1200MB'
slurm_information['email'] = 'yourslurmnotificationemailaddress@gmail.com'
slurm_information['vasp_version'] = 'VASP/5.3.5-intel-2017a-VTST-BEEF'
slurm_information['vasp_execution'] = 'vasp_cd'
```

(continues on next page)

(continued from previous page)

```

# ---------------------------------------------------------------------------------------------------------------------------
# Run the Adsorber program
Adsorber_Program(part_to_perform,cluster_or_surface_model,system_filename,path_to_VASP_optimised_non_adsorbate_system,cutoff,surface_atoms,adsorbed_species,slurm_information,Other_molecules_to_obtain_VASP_energies_for)
```

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

1) Things to import into this script

First you will want to import the `Adsorber` program, as well as any other methods that you want to use to import atoms and molecules to adsorb upon your cluster or surface. Here, we have imported the `molecule` method from the `ase.build` module.

```python
from ase.build import molecule
from Adsorber import Adsorber_Program
```

2) Initial inputs for `Adsorber`

To begin, there are six inputs you will need to give to Adsorber. These are:

- **part_to_perform** (*str.*): This variable indicates which Part of of Adsorber protocol you would like to perform. This is either `'Part A'`, `'Part B'`, or `'Part C'`.
- **cluster_or_surface_model** (*str.*): This tells `Adsorber` if you are wanting to adsorb atoms and clusters to the surface of a cluster or a surface model. If you are dealing with a cluster, set `cluster_or_surface_model = 'cluster'`, else if you are dealing with a surface model, set `cluster_or_surface_model = 'surface model'` and make sure that your surface and vacuum point in the positive z direction. Also if you want to constrain any atoms in your model, this should be done in this file.

Part A options:

- **system_filename** (*str.*): The name of the file of the cluster or the surface model that you will like to import into Adsorber. This file should be a `.xyz` or `.traj` file, but in reality any file type will do that ASE can read (see https://wiki.fysik.dtu.dk/ase/ase/io/io.html for more information on formats that ASE can read). For surfaces, you should have already performed surface convergence studies before proceeding with part A.

Part B options:

- **path_to_VASP_optimised_non_adsorbate_system** (*str.*): This is the directory to your surface or cluster model without adsorbate after you have optimised it with VASP.
  - For clusters, this should be set to `path_to_VASP_optimised_non_adsorbate_system = 'Part_A_Non_Adsorbed_Files_For_VASP/system/OUTCAR'`
  - For surfaces, this should be pointed to the file that you want to bind your adsorbate to. This may be `path_to_VASP_optimised_non_adsorbate_system = 'Part_A_Non_Adsorbed_Files_For_VASP/system/OUTCAR'`, or if you have surface converged your surface model, then to the file that represents the surface converged model.
- **cutoff** (*float* or *dict.*): This is the maximum distance between atoms to be considered `bonded` or `neighbouring`. This is used to determine bridging, three-fold, and four-fold sites. This is given as a float for monoatomic cluster and surface systems, or for a multiatomic system if you are happy for the max bonding

distance between any two elements to be the same. If you would like different element pairs to have different maximum bonding distances, this is given as a dictionary. For example, for a CuPd system: `cutoff = {'Cu': 3.2, 'Pd': 3.6, ('Cu','Pd'): 3.4}`

- **surface_atoms** (*list of ints*): This is a list of the indices of all the surface atoms in your cluster or surface model. See *How to Mark Surface Atoms in your Cluster/Surface model in Adsorber* for how to determine which of your atoms are surface atoms and to get those clusters indices to add to the `surface_atoms` list. Note that if there are surface atoms that you do not want molecules to adsorb to, dont include them in this list.

This is given in this example as below:

```
# Initial inputs for the Adsorber program
part_to_perform = 'Part B'

# General Variables
cluster_or_surface_model = 'cluster'

# Part A information
system_filename = '15-3-3629_with_vacuum_6.0_Ang.xyz'

# Part B information
path_to_VASP_optimised_non_adsorbate_system = 'Part_A_Non_Adsorbed_Files_For_VASP/
↪system/OUTCAR'
cutoff = 3.5
surface_atoms = [11,25,28,13,3,8,6,23,22,59,34,62,66,1,0,4,30,15,14,16,5,12,29,2,7,10,
↪24,26,70,35,47,50,60,63,48,39,41,44,54,68,76,71,32,31,74,42,56,52,43,40,46,61,53,45,
↪57,72,73,77]
```

3) Add the Atoms and Molecules on to the Surface of your Cluster/Surface Model

We will now add all the atoms and molecules that you want to adsorb to the surface of your cluster or surface model. We first want to import all of these into this script. In ASE, there are a variety of molecules you can obtain from the `molecule` method in the `ase.build` module. This is imported into the `Run_Adsorber.py` script with the following

```
from ase.build import molecule
```

This is what we have done here. You can also import molecules from other `.xyz` or `.traj` files with the `read` method from ASE:

```
from ase.io import read
```

For each atom and molecule that you make you want to add it to a dictionary that has the following inputs:

- **name** (*str.*): The name you want to give for this atom or molecule
- **molecule** (*ase.Atoms*): The is the `Atoms` object for the atom or molecule, as obtained from the `molecule` or `read` method as mentioned above.
- **distance_of_adatom_from_surface** (*float*): This is the binding distance that you would like the atom or molecule to be initially placed from the cluster or surface model before you perform further optimisation with DFT.

For single atoms, this is all that is needed. If you want to adsorb molecules that have two or more atoms in it, you want to give two or three additional inputs into this dictionary.

- **index** (*int.*): This is the index of the atom in the molecule to adsorb to the surface for the cluster/surface model. See *How to Bind Molecule to the Surface of your Cluster/Surface Model in Adsorber* for more information on how to select the index of the atom in the molecule you would like to be adsorbed to the surface.

- **axis** (*str./list/tuple*): This is the axis in your molecule that you would like to point away from the surface of the cluster/surface model, as well as to rotate your moleule around (if you would like to rotate your molecule around the axis). See *How to Bind Molecule to the Surface of your Cluster/Surface Model in Adsorber* for more information for how to specify this axis.
- **rotations** (*list/tuple*, optional): These are the angles of rotation that you would like to rotate the molecules around the **axis** on the surface of your cluster/surface model. If you have a linear molecule that is alligned to the **axis** or you do not want to rotate your molecule around the **axis**, you do not need to add this as this is an optional input. See *How to Bind Molecule to the Surface of your Cluster/Surface Model in Adsorber* for more information about how to specify how to best rotate your molecule about the **axis** on the surface of your cluster/surface model.

An example of this is shown below:

```
20 # Give the Atoms objects for the atoms and molecules you want to adsorb to your cluster or surface model
21 adsorbed_species = []
22
23 COOH_symmetric = molecule('HCOOH') # note the carbon is index 1
24 COOH_symmetric.center(vacuum=10.0)
25 del COOH_symmetric[4] # remove the hydrogen atom
26 COOH_symmetric_axis = (0.1,-1,0)
27 distance_of_adatom_from_surface = 1.5
28 rotations = 'automatic' #range(0,360,10)
29 COOH_symmetric_adsorbed_species = {'name': 'COOH_symmetric', 'molecule': COOH_symmetric, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 1, 'axis': COOH_symmetric_axis, 'rotations': rotations, 'sites_to_bind_adsorbate_to': ['Top_Sites','Bridge_Sites','Three_Fold_Sites']}
30 adsorbed_species.append(COOH_symmetric_adsorbed_species)
```

You want to make sure that you append each of your adsorbates to the `adsorbed_species` list. If you dont want to include certain adsorbates as you go about your studies, comment their `adsorbed_species.append(...)` line. For example, see line 39 below:

```
32 COOH_O_tilted = molecule('HCOOH') # note the carbon is index 1
33 COOH_O_tilted.center(vacuum=10.0)
34 del COOH_O_tilted[4] # remove the hydrogen atom
35 COOH_O_tilted_axis = (-0.4,-1,0)
36 distance_of_adatom_from_surface = 1.5
37 rotations = 'automatic' #range(0,360,10)
38 COOH_O_tilted_adsorbed_species = {'name': 'COOH_O_tilted', 'molecule': COOH_O_tilted, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 1, 'axis': COOH_O_tilted_axis, 'rotations': rotations, 'sites_to_bind_adsorbate_to': 'Top_Sites'}
39 #adsorbed_species.append(COOH_O_tilted_adsorbed_species)
```

The full example of the atoms and molecules that have been adsorbed to this cluster (called `15-3-3629.xyz`) is shown below:

```
20 # Give the Atoms objects for the atoms and molecules you want to adsorb to your cluster or surface model
21 adsorbed_species = []
22
23 COOH_symmetric = molecule('HCOOH') # note the carbon is index 1
24 COOH_symmetric.center(vacuum=10.0)
25 del COOH_symmetric[4] # remove the hydrogen atom
26 COOH_symmetric_axis = (0.1,-1,0)
27 distance_of_adatom_from_surface = 1.5
```

(continues on next page)

(continued from previous page)

```
rotations = 'automatic' #range(0,360,10)
COOH_symmetric_adsorbed_species = {'name': 'COOH_symmetric', 'molecule': COOH_symmetric, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 1, 'axis': COOH_symmetric_axis, 'rotations': rotations, 'sites_to_bind_adsorbate_to': ['Top_Sites','Bridge_Sites','Three_Fold_Sites']}
adsorbed_species.append(COOH_symmetric_adsorbed_species)

COOH_O_tilted = molecule('HCOOH') # note the carbon is index 1
COOH_O_tilted.center(vacuum=10.0)
del COOH_O_tilted[4] # remove the hydrogen atom
COOH_O_tilted_axis = (-0.4,-1,0)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
COOH_O_tilted_adsorbed_species = {'name': 'COOH_O_tilted', 'molecule': COOH_O_tilted, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 1, 'axis': COOH_O_tilted_axis, 'rotations': rotations, 'sites_to_bind_adsorbate_to': 'Top_Sites'}
#adsorbed_species.append(COOH_O_tilted_adsorbed_species)

CO = molecule('CO') # note the carbon is index 1
CO.center(vacuum=10.0)
CO_axis = 'z'
distance_of_adatom_from_surface = 1.5
CO_adsorbed_species = {'name': 'CO', 'molecule': CO, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 1, 'axis': CO_axis}
#adsorbed_species.append(CO_adsorbed_species)

# --------------------------------------------
# option1
COH = molecule('H2COH') # note the carbon is index 0
COH.center(vacuum=10.0)
del COH[4] # remove the hydrogen atom
del COH[3] # remove the hydrogen atom
COH_axis = '-x'
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
COH_adsorbed_species = {'name': 'COH', 'molecule': COH, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 0, 'axis': COH_axis, 'rotations': rotations}
#adsorbed_species.append(COH_adsorbed_species)

#option2
CHO = molecule('HCO') # note the carbon is index 0
CHO.center(vacuum=10.0)
CHO_axis = (-(3.0**0.5)/2.0,-1.0/2.0,0)
rotations = 'automatic' #range(0,360,10)
CHO_adsorbed_species = {'name': 'CHO', 'molecule': CHO, 'distance_of_adatom_from_surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CHO_axis, 'rotations': rotations}
#adsorbed_species.append(CHO_adsorbed_species)
# --------------------------------------------
# option1
CH2O = molecule('H2CO') # note the carbon is index 1
CH2O.center(vacuum=10.0)
CH2O_axis = 'x'
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
```

(continues on next page)

(continued from previous page)

```
CH2O_adsorbed_species = {'name': 'CH2O', 'molecule': CH2O, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 1, 'axis': CH2O_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CH2O_adsorbed_species)

# option1
CHOH = molecule('H2COH') # note the carbon is index 0
CHOH.center(vacuum=10.0)
del CHOH[3]
CHOH_axis = (-1,-1,0)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
CHOH_adsorbed_species = {'name': 'CHOH', 'molecule': CHOH, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CHOH_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CHOH_adsorbed_species)
# ---------------------------------------------
# new option 1
CH3O = molecule('CH3O') # note the oxygen is index 1
CH3O.center(vacuum=10.0)
CH3O_axis = '-y'
distance_of_adatom_from_surface = 1.5
CH3O_adsorbed_species = {'name': 'CH3O', 'molecule': CH3O, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 1, 'axis': CH3O_axis}
#adsorbed_species.append(CH3O_adsorbed_species)

# new option 2
CH2OH = molecule('CH3OH') # carbon is index 0
CH2OH.center(vacuum=10.0)
del CH2OH[2] # remove the hydrogen atom
CH2OH_axis = (1,-1,0)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,360,10)
CH2OH_adsorbed_species = {'name': 'CH2OH', 'molecule': CH2OH, 'distance_of_adatom_
↪from_surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CH2OH_axis,
↪'rotations': rotations}
#adsorbed_species.append(CH2OH_adsorbed_species)
# ---------------------------------------------
# new new option 1
CH2 = molecule('CH4') # carbon is index 0
CH2.center(vacuum=10.0)
del CH2[4] # remove the hydrogen atom
del CH2[3] # remove the hydrogen atom
CH2_axis = 'z'
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,180,10)
CH2_adsorbed_species = {'name': 'CH2', 'molecule': CH2, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CH2_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CH2_adsorbed_species)

CH3 = molecule('CH4') # carbon is index 0
CH3.center(vacuum=10.0)
del CH3[4] # remove the hydrogen atom
CH3_axis = (1,-1,1)
distance_of_adatom_from_surface = 1.5
rotations = 'automatic' #range(0,120,10)
```

(continues on next page)

(continued from previous page)

```
CH3_adsorbed_species = {'name': 'CH3', 'molecule': CH3, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 0, 'axis': CH3_axis, 'rotations
↪': rotations}
#adsorbed_species.append(CH3_adsorbed_species)

# new new option 2
O = molecule('O')
O.center(vacuum=10.0)
distance_of_adatom_from_surface = 1.5
O_adsorbed_species = {'name': 'O', 'molecule': O, 'distance_of_adatom_from_surface':␣
↪distance_of_adatom_from_surface}
#adsorbed_species.append(O_adsorbed_species)

OH = molecule('OH') # note the oxygen is index 0
OH.center(vacuum=10.0)
OH_axis = '-z'
distance_of_adatom_from_surface = 1.5
OH_adsorbed_species = {'name': 'OH', 'molecule': OH, 'distance_of_adatom_from_surface
↪': distance_of_adatom_from_surface, 'index': 0, 'axis': OH_axis}
#adsorbed_species.append(OH_adsorbed_species)

C = molecule('C')
C.center(vacuum=10.0)
distance_of_adatom_from_surface = 1.5
C_adsorbed_species = {'name': 'C', 'molecule': C, 'distance_of_adatom_from_surface':␣
↪distance_of_adatom_from_surface}
#adsorbed_species.append(C_adsorbed_species)
```

4) Obtain VASP energies for molecules that you do not want to adsorb to your cluster/surface model

You made also want to obtain locally optimised VASP energies of atoms and molecules for energy calculations later on in your studies. To do this, you want to add your atoms and molecules in the same way as before, such as with `xyz` files or using the `molecule` method in the `ase.build` module, as mentioned in *3) Add the Atoms and Molecules on to the Surface of your Cluster/Surface Model*. For each atom and molecule that you make you want to add it to a dictionary that has the following inputs:

- **name** (*str.*): The name you want to give for this atom or molecule
- **molecule** (*ase.Atoms*): The is the `Atoms` object for the atom or molecule, as obtained from the `molecule` or `read` method as mentioned above.

Example of this are shown below for obtaining VASP minimised energies only of graphene, H_2, and H_2O:

```
# Add here any other atoms and molecules that you need to locally optimise in VASP␣
↪for energy calculations
Other_molecules_to_obtain_VASP_energies_for = []

graphene = molecule('C')
graphene.center(vacuum=10.0)
graphene_optimise_energy = {'name': 'graphene', 'molecule': graphene}
Other_molecules_to_obtain_VASP_energies_for.append(graphene_optimise_energy)

H2 = molecule('H2')
H2.center(vacuum=10.0)
H2_optimised_energy = {'name': 'H2', 'molecule': H2}
Other_molecules_to_obtain_VASP_energies_for.append(H2_optimised_energy)
```

(continues on next page)

(continued from previous page)

```

H2O = molecule('H2O')
H2O.center(vacuum=10.0)
H2O_optimised_energy = {'name': 'H2O', 'molecule': H2O}
Other_molecules_to_obtain_VASP_energies_for.append(H2O_optimised_energy)
```

5) Information required to make `submit.sl` siles for submitting files to Slurm

`Adsorber` is able to create folders with the files required to run VASP jobs of your system for all of the adsorbed species and orientations that you would like to consider. Do you this, you will want to include a dictionary called `slurm_information` that contains all the information about the `submit.sl` file required to submit jobs to the Slurm workload Manager (https://slurm.schedmd.com/documentation.html). The following information is required in the `slurm_information` dictionary:

- `'project'` (*str.*): The name of the project to run this on.
- `'partition'` (*str.*): The partition to run this on.
- `'time'` (*str.*): The length of time to give these jobs. This is given in 'HH:MM:SS', where HH is the number of hours, MM is the number of minutes, and SS is the number of seconds to run `Adsorber` for.
- `'nodes'` (*int*): The number of nodes to use. Best to set this to 1.
- `'ntasks_per_node'` (*int*): The number of cpus to run these jobs across on a node for a VASP job.
- `'mem-per-cpu'` (*str.*): This is the memory that is used per cpu by the job.
- `'email'` (*str.*): This is the email address to send slurm messages to about this job. If you do not want to give an email, write here either `None` or `''`.
- `'vasp_version'` (*str.*): This is the version of VASP that you want to use for your VASP job. Default: `'VASP/5.4.4-intel-2017a'`
- `'vasp_execution'` (*str.*): This is the command that is required to run a VASP job. Default: `'vasp_std'`

An example of a `slurm_information` dictionary in the `Run_Adsorber.py` script is shown below:

```
# slurm informaion for making the submit.sl files for submitting VASP jobs in slurm

slurm_information = {}
slurm_information['project'] = 'uoo02568'
slurm_information['partition'] = 'large'
slurm_information['time'] = '72:00:00'
slurm_information['nodes'] = 1
slurm_information['ntasks_per_node'] = 12
slurm_information['mem-per-cpu'] = '1200MB'
slurm_information['email'] = 'yourslurmnotificationemailaddress@gmail.com'
slurm_information['vasp_version'] = 'VASP/5.3.5-intel-2017a-VTST-BEEF'
slurm_information['vasp_execution'] = 'vasp_cd'
```

Run the Adsorber Program!

You have got to the end of all the parameter setting stuff! Now on to the fun stuff! The next part of the Run_Adsorber.py file will run the Adsorber program. This is written as follows in the Run_Adsorber.py script:

```
# Run the Adsorber program
Adsorber_Program(part_to_perform,cluster_or_surface_model,system_filename,path_to_
↪VASP_optimised_non_adsorbate_system,cutoff,surface_atoms,adsorbed_species,slurm_
↪information,Other_molecules_to_obtain_VASP_energies_for)
```

4.5 Prelude 2: How to obtain some of the settings for the Run_Adsorber.py script

Following the *Prelude 1: Run_Adsorber.py - Running Adsorber* page that indicated what settings and inputs are needed to run the Run_Adsorber.py script, on this webpage we will describe how to obtain some of the settings that are not obvious as to how to obtain them from the *Prelude 1: Run_Adsorber.py - Running Adsorber* page. This includes how to obtain the information required to tell Adsorber how to bind atoms and molecules to the surface of your system.

This page requires the use of ASE GUI to view our chemical systems. The installation and use of this visualisation programs is given in *External Programs that will be useful to install for using Adsorber*.

4.5.1 How to Mark Surface Atoms in your Cluster/Surface model in Adsorber

One of the pieces of information that Adsorber need to know are which atoms on your cluster are the surface atoms. The easiest way to figure out which atoms are surface atoms is to open your cluster/surface model in ASE GUI. An example is given below:

Then you want to go to show the indices of the atoms in your cluster/surface model by clicking in the menu View > Show Labels > Atom Index. This will show the indices of atoms in your cluster/surface model in your ASE GUI.

We will want to include the indices of the surface atoms in your cluster/surface model in the Run_Adsorber.py file in the surface_atoms list. In the example given in the Examples/Cu78_Example/15-3-3629.xyz file, the surface atoms are:

```
surface_atoms = [11,25,28,13,3,8,6,23,22,59,34,62,66,1,0,4,30,15,14,16,5,12,29,2,7,10,
↪24,26,70,35,47,50,60,63,48,39,41,44,54,68,76,71,32,31,74,42,56,52,43,40,46,61,53,45,
↪57,72,73,77]
```

The Adsorber program will create a .xyz file called SYSTEM_NAME_tagged_surface_atoms.xyz that will have all surface atoms tagged 1 and all non-surface atoms tagged 0 (where SYSTEM_NAME is the name of the .xyz or .traj file that you gave for the name variable in Run_Adsorber.py, see *Prelude 1: Run_Adsorber.py - Running Adsorber*). You can see this if you open SYSTEM_NAME_tagged_surface_atoms.xyz in ASE GUI, show atom index label by clicking View > Show Labels > Atom Index, and colouring in atoms based on their tag by clicking View > Colors and selecting By tag:

All the surface atoms should be coloured pink, while the non-surface atoms coloured green. If there are any bulk atom coloured pink or surface atoms coloured green, you will need to remove or add the indices of atoms from the surface_atoms list in the Run_Adsorber.py file to make sure that surface_atoms reflects the atoms in the cluster/surface model that are in fact surface atoms.

Note that if there are surface atoms that you do not want molecules to adsorb to, dont include them in this list.

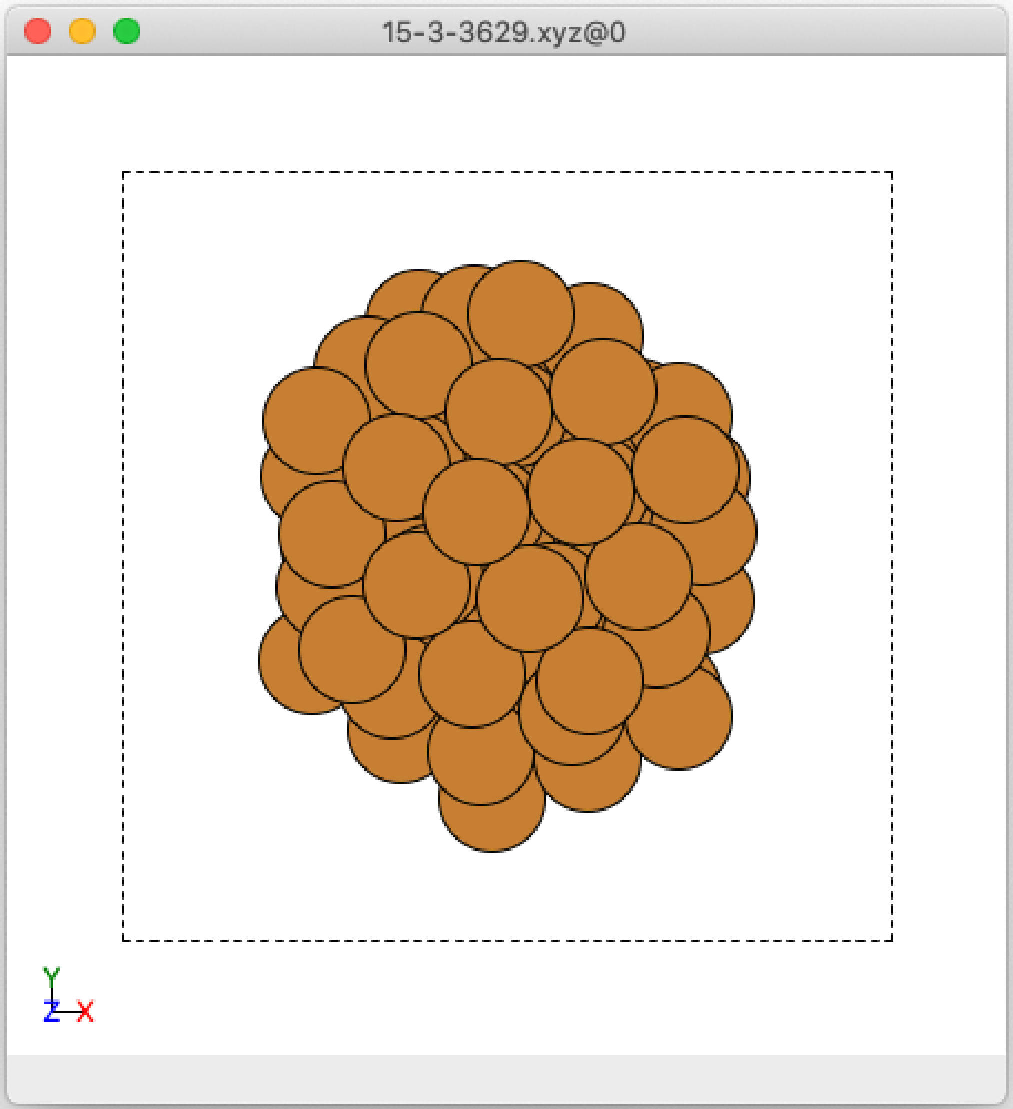


Fig. 6: An example cluster when observed in ASE GUI.

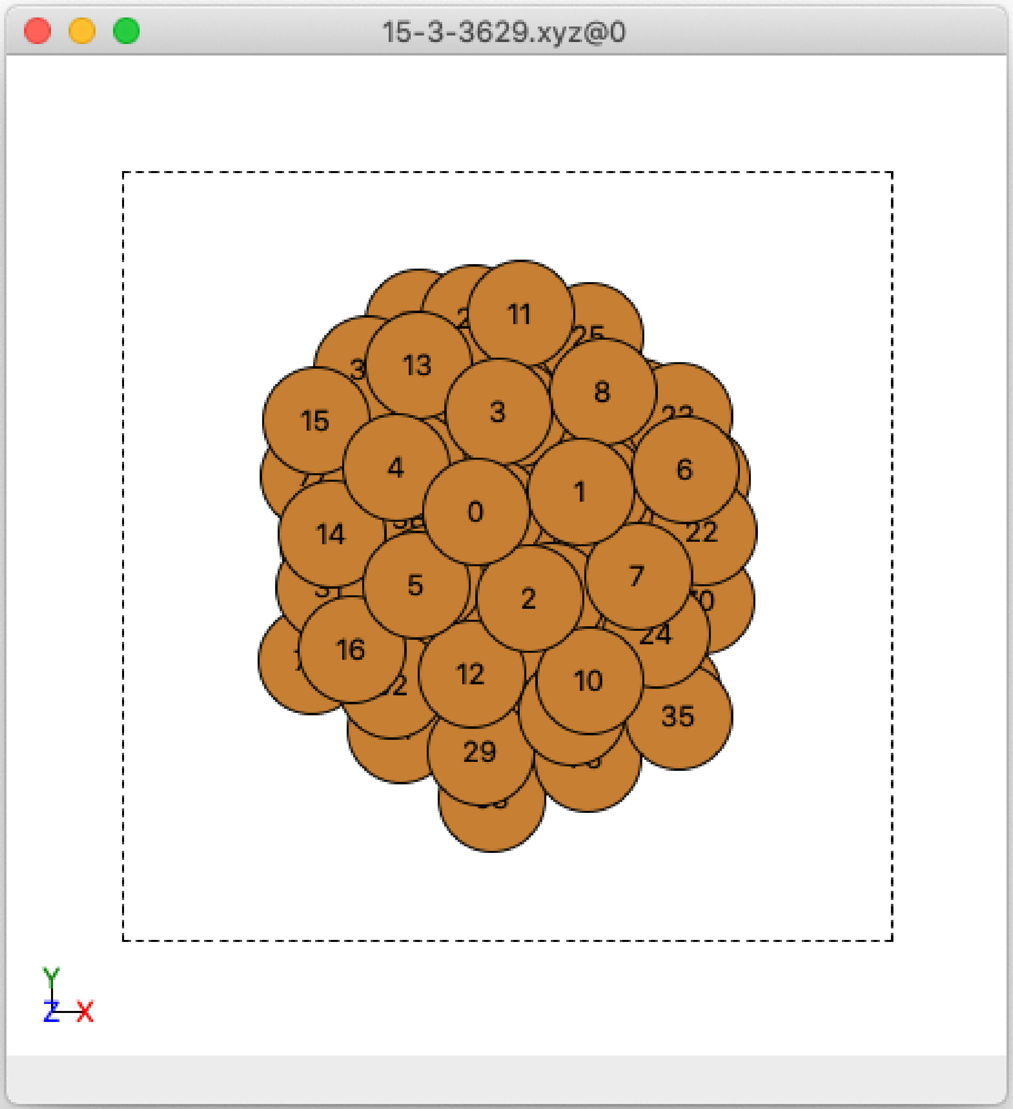


Fig. 7: An example cluster when observed in ASE GUI, where atoms have been labelled by their indices.

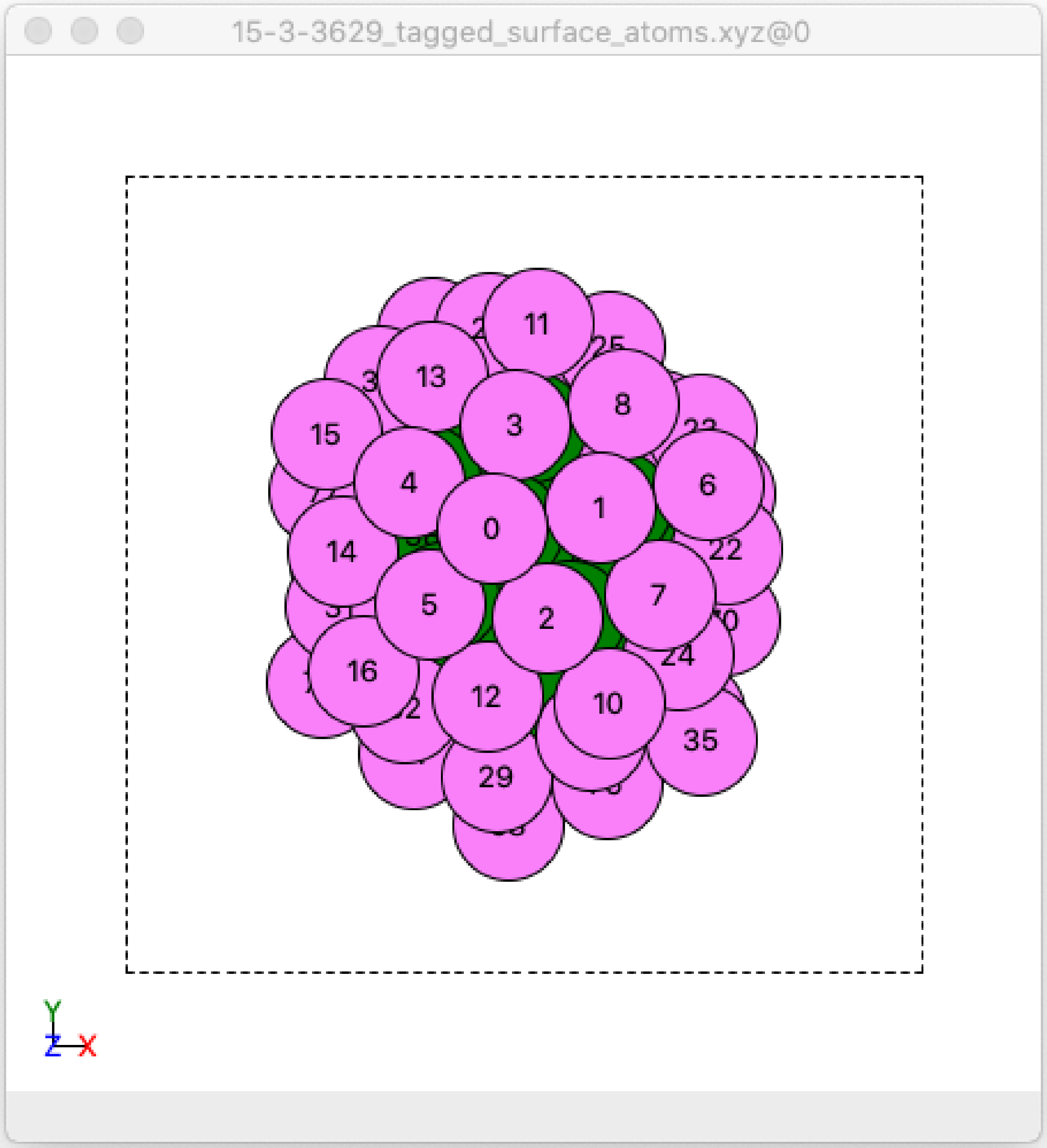


Fig. 8: Front view of an image of the example cluster where surface atoms are coloured pick, and non-surface atoms coloured green. This example `.xyz` file is created by `Adsorber`.

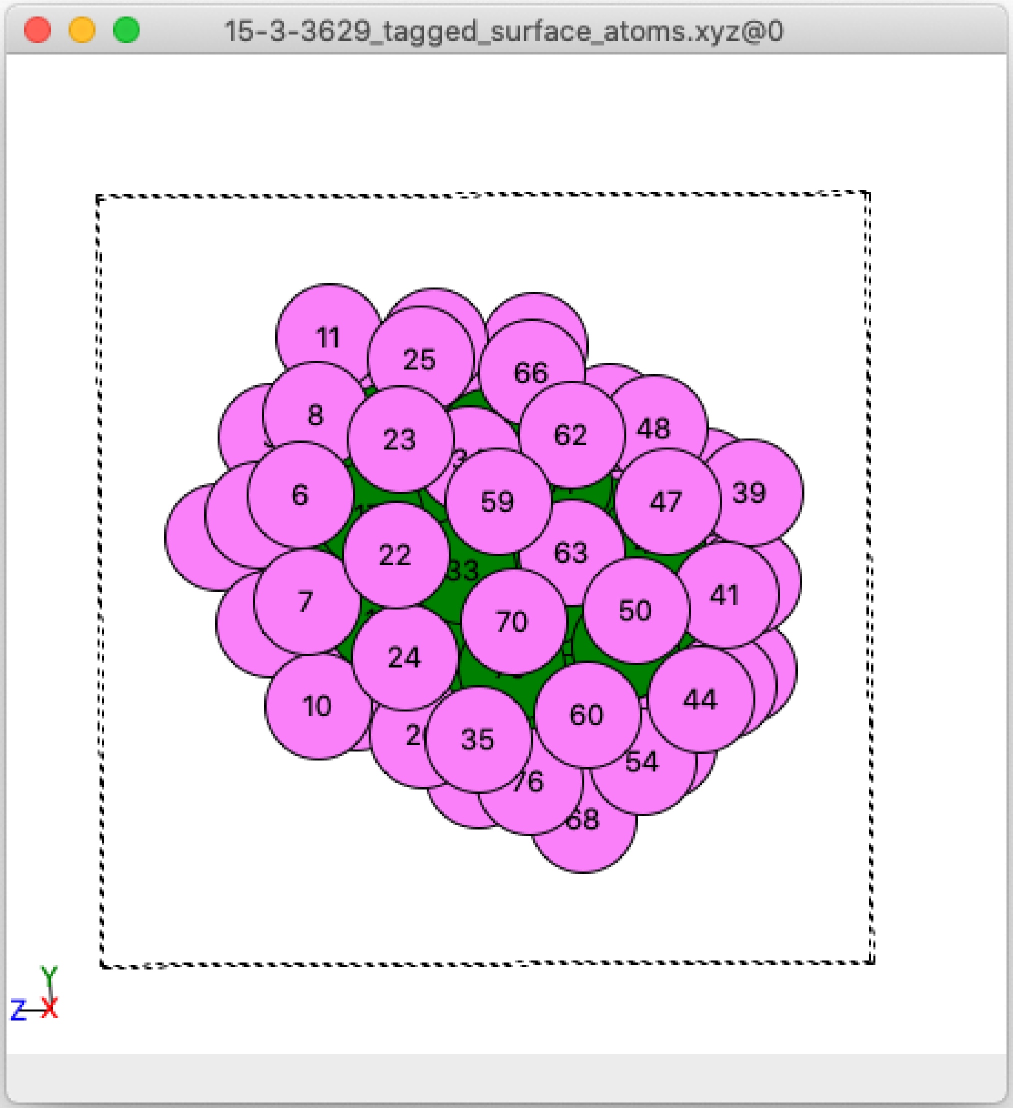


Fig. 9: Side view of an image of the example cluster where surface atoms are coloured pick, and non-surface atoms coloured green. This example `.xyz` file is created by `Adsorber`.

4.5.2 How to Bind Molecule to the Surface of your Cluster/Surface Model in Adsorber

There are many ways that a molecule can be bound to the surface of a cluster/surface model. There are two or three components that are important for adsorbing a molecule to a surface. These are the atom in the molecule that is bound to the surface *via* (**index**), the alignment of the molecule to the surface (**axis**), and the orientation of the molecule to the surface (**rotations**). These are given by the following three entries in the `adsorbed_species` dictionary for each molecule.

- **index** (*int.*): This is the index of the atom in the molecule to adsorb to the surface for the cluster/surface model.
- **axis** (*str./list/tuple*): This is the axis in your molecule that you would like to point away from the surface of the cluster/surface model, as well as to rotate your moleule around (if you would like to rotate your molecule around the axis).
- **rotations** (*list/tuple*, optional): These are the angles of rotation that you would like to rotate the molecules around the **axis** on the surface of your cluster/surface model. If you have a linear molecule that is alligned to the **axis** or you do not want to rotate your molecule around the **axis**, you do not need to add this as this is an optional input.

We will now describe the way to determine the **index**, **axis**, and **rotations** variables.

Specifying the index variable

First, open up the molecule that you want to adsorb to the surface in ASE. You can do this using the ASE GUI. If you are making your molecules in your `Run_Adsorber.py` you can take a look at it using the `view` method in the `ase.visualize` module:

```python
from ase.visualize import view
from ase.build import molecule

COOH = molecule('HCOOH') # note the carbon is index 1
del COOH[4] # remove the hydrogen atom

view(COOH)
```

This will open a ASE GUI of the COOH molecule. If you click in the main menu `View > Show Labels > Atom Index`, you will get the following view:

We would like to adsorb the C atom in the COOH molecule to the surface of our cluster/surface model. In this case, we would like to set **index** for this molecule to `1` to specify index 1.

Specifying the axis variable

Next, we would like to specify the direction of the axis we would like to align our molecule to the surface of your cluster/surface model.

This will align this axis to the “normal” of the surface. For above atom and bridge site, this axis will point as far away from any other atom on the cluster/surface model as possible. For this reason, we will point this vector from the **index** atom, which in this example is the index 1 C atom in this COOH molecule.

For this example, we would like to align the molecule on the surface such that the oxygen atoms are on opposite sides of the **axis** vector as possible. THis can be achieved with a vector that points `(0.1,-1,0)` from looking at the above figure of the COOH atom. The `Adsorber` program will use the Rodrigues formula[15] to rotate the molecule by this axis onto the “normal” vector of the surface.

[15] https://en.wikipedia.org/wiki/Rodrigues%27_formula

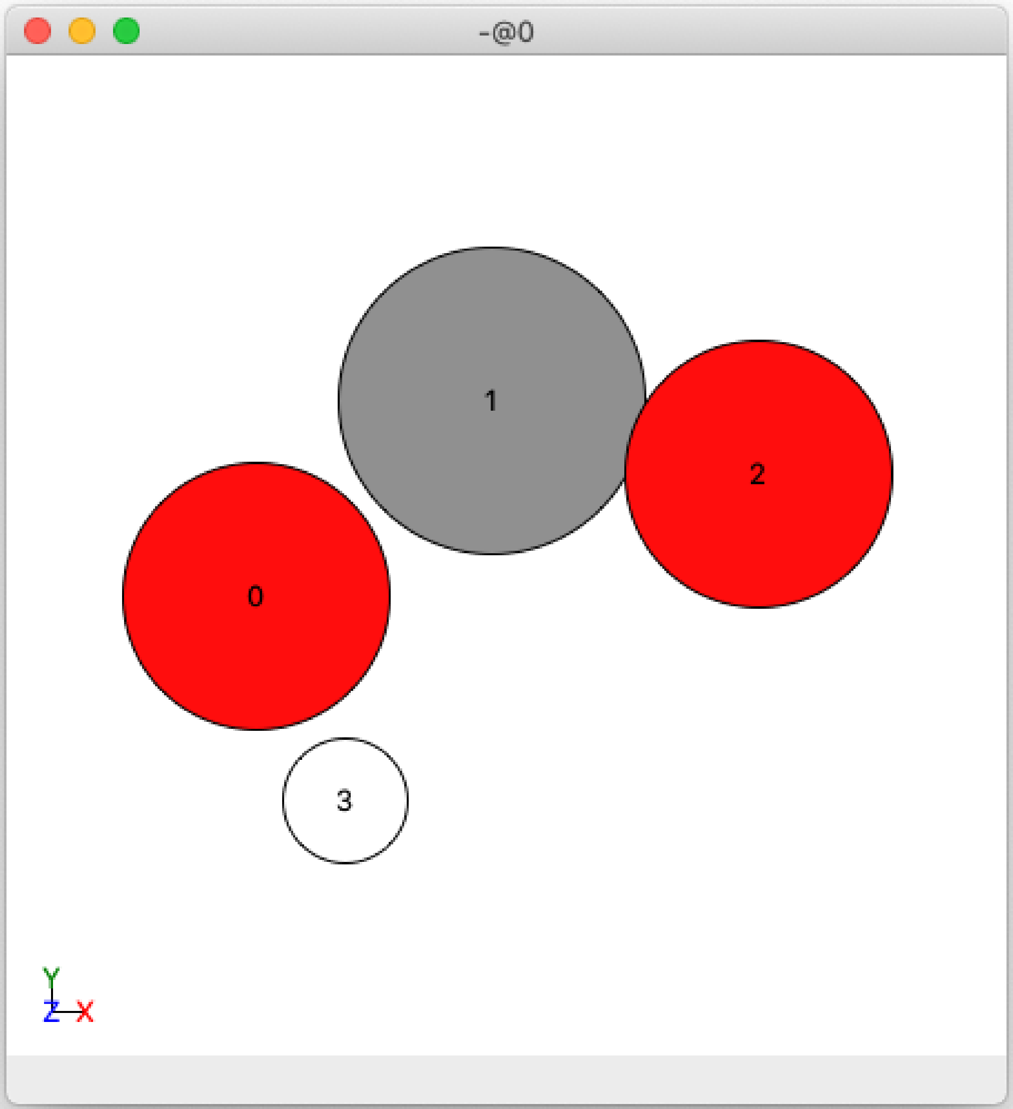


Fig. 10: View of a COOH molecule in ASE GUI, where the atoms have been labelled by their index.

Note that you can also specify the following string for the **axis** vector:

- `'x'`: This is the `(1,0,0)` vector
- `'y'`: This is the `(0,1,0)` vector
- `'z'`: This is the `(0,0,1)` vector
- `'-x'`: This is the `(-1,0,0)` vector
- `'-y'`: This is the `(0,-1,0)` vector
- `'-z'`: This is the `(0,0,-1)` vector

Specifying the rotations variable

Often you will want to sample a specific orientation for the molecule to adsorb to the surface of the cluster/surface model. `Adsorber` allows you to rotate the ad-molecule about the **axis** vector on the surface of your cluster/surface model to try to get the orientation(s) that you like.

`Adsorber` has various options for how to rotate your adsorbate about each binding site on your system. You can choose to allow `Adsorber` to automatically choose how to rotate the adsorbate about each binding site. To choose this, set *rotation = 'automatic'*:

```
Adsorbed_Species['rotations'] = 'automatic'
```

`Adsorber` will rotate the adsorbate so that each atom in the adsorbate is alligned to each nearby surface atom in the system, as well as alligned between those nearby surface atoms. However, some of the rotations may allign the adsorbate onto a local maximum that VASP does not optimise out of during a VASP local optimisation. For this reason, you can also specify `Adsorber` to slightly misalign each rotation by $\pm$ XX degrees:

```
Adsorbed_Species['rotations'] = 'automatic with misalignment of XX degrees'
```

where XX is the rotation in degrees to misalign the adsorbates rotation by. For example, if you want to misalign an adsorbate by 10 degrees:

```
Adsorbed_Species['rotations'] = 'automatic with misalignment of 10 degrees'
```

You can also choose custom rotations to rotate your adsorbate by. You can choose this option by specifying the angles you would like to rotate your adsorbate by in a list. For example:

```
Adsorbed_Species['rotations'] = range(0,360,10)
```

If you choose custom rotations, it is recommended to try a wide range of rotations and delete those rotations that do not work for you. To do this, set rotation to a range of angles, such as `range(0,360,10)`, where the angles are given in degrees. This will create 36 `.xyz` files of the same molecule adsorbed to the same site on the cluster/surface feature, where each image is a different rotated orientation of the molecule on the surface of your cluster/surface feature.

The youtube clip below shows an example of all the orientations that are created by the `Adsorber` program for a COOH molecule adsorbed to a vertex site on a Cu78 nanocluster made by `Adsorber` (where `Adsorbed_Species['rotations'] = range(0,360,10)`).

How to enter index, axis, and rotations into the `adsorbed_species` dictionary in the `Run_Adsorber.py` script

Once you have specied the **index**, **axis**, and **rotations**, you can add them to the `adsorbed_species` dictionary in the `Run_Adsorber.py` script. An example for a COOH molecule is shown below:

```
COOH = molecule('HCOOH') # note the carbon is index 1
del COOH[4] # remove the hydrogen atom
COOH_axis = (0.1,-1,0)
distance_of_adatom_from_surface = 1.25
rotations = range(0,360,10)
COOH_adsorbed_species = {'name': 'COOH', 'molecule': COOH, 'distance_of_adatom_from_
↪surface': distance_of_adatom_from_surface, 'index': 1, 'axis': COOH_axis, 'rotations
↪': rotations}
```

This dictionary is then appended to the `adsorbed_species` list in the `Run_Adsorber.py` script. See *Add the Atoms and Molecules on to the surface of your Cluster/Surface Model*.

4.6 Part A: How to optimise your system initially

To begin, we need to obtain the locally optimise version of the system you want to adsorb adsorbates onto. To do this, you will want to **set the** `Step_to_Perform` **variable in the** `Run_Adsorber.py` **script to** `'Part A'`:

```
Step_to_Perform = 'Part A'
```

You also need to create a folder called `VASP_Files` that contains the following files and folders:

- `INCAR`: This is a VASP file that contains all the information required to run the VASP job (https://www.vasp.at/wiki/index.php/INCAR and https://cms.mpi.univie.ac.at/vasp/vasp/INCAR_File.html).
- `KPOINTS`: The KPOINTS file is used to specify the Bloch vectors (k-points) that will be used to sample the Brillouin zone in your calculation (https://www.vasp.at/wiki/index.php/KPOINTS).
- `POTCARs`: This is a folder that contains all of the `POTCAR` files for all of the different elements in your models. Each of the `POTCAR` files in this folder need to be labelled as `POTCAR_XX`, where `XX` is the symbol for the particular element. For example, for the POTCAR to describe Cu, you want to name the POTCAR as `POTCAR_Cu`, the POTCAR for C should be called `POTCAR_C`, the POTCAR for H should be called `POTCAR_H`, ...

You want to also include any other files that will be needed. For example, if you are running VASP with the BEEF functional, you need to include the `vdw_kernel.bindat` file in the `VASP_Files` folder.

An example of `VASP_Files` folders can be found in Adsorber Examples on Github[16].

You can then run the `Run_Adsorber.py` script in the terminal:

```
python Run_Adsorber.py
```

This will create a folder called `Part_A_Non_Adsorbed_Files_For_VASP`. In this folder is another folder called `system` that contains all the VASP files required to locally optimise your system in VASP.

This `Part_A_Non_Adsorbed_Files_For_VASP` folder also contains all the VASP files of your adsorbates that you can also locally optimise in VASP. This is often required to determine in the adsorption energy of your adsorbate onto your system

$$E_{abs} = E(system + adsorbate) - (E(system) + E(adsorbate))$$

[16] https://github.com/GardenGroupUO/Adsorber/tree/main/Example

or if you are defining your adsorption energy with reference species, such as C from graphene, H from H_2 , and O from H_2O:

$$E_{abs} = E(system + adsorbate) - (E(system) + E(each\ element\ in\ the\ adsorbate\ in\ stiochiometic\ amounts))$$

Once you have run VASP on your system and all your adsorbates, proceed to Part B (*Part B: Adsorbing Adsorbates to your System*).

4.7 Part B: Adsorbing Adsorbates to your System

Once you have locally optimsed your system, you can adsorb your adsorbates to your system. To do this, **set the** `Step_to_Perform` **variable in the** `Run_Adsorber.py` **script to** `'Part B'`:

```
Step_to_Perform = 'Part B'
```

You can then run the `Run_Adsorber.py` script in the terminal:

```
python Run_Adsorber.py
```

This will create a folder called `Part_B_All_Systems_with_Adsorbed_Species` that contains adsorbates that are adsorbed at various sites across your system in `xyz` format. `xyz` files are found in the path: `Part_B_All_Systems_with_Adsorbed_Species\ADSORBATE\ADSORPTIONSITE`, where `ADSORBATE` is the adsorbate you want to focus on, and `ADSORPTIONSITE` is the type of surface site that the adsorbate is bound to, being:

- `Top_Sites`: These are adsorbates that are bound to top sites above each surface atom.
- `Bridge_Sites`: These are adsorbates that are bound to bridging sites.
- `Three_Fold_Sites`: These are adsorbates that are bound to three fold sites.
- `Four_Fold_Sites`: These are adsorbates that are bound to four fold sites.

Adsorber will created many `xyz` files, many of which you may not want to run in VASP. This may be because many of the sites are structural degenerate, or to orientate the adsorbate in certain directions.

In the next step, we choose which of these `xyz` files to further locally optimise in VASP. To do this, VASP will also create another folder called `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files`. What you want do is to **choose is to select the adsorbates in the sites and orientations that you want to optimise in VASP and copy them into the** `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files` **folder**.

4.7.1 How to Choose Which `xyz` Files to Optimise in VASP

We will now discuss how to choose `xyz` files for further optimise in VASP. This section requires the use of the ASE GUI and Jmol to view our chemical systems. The installation and use of these visualisation programs is given in *External Programs that will be useful to install for using Adsorber*.

To help determine which surface sites to adsorb adsorbates to, `Adsorber` will create a folder called `Part_B_Binding_Site_Locations` that contains four xyz files that show all the binding sites found in your system. These are:

1. `SYSTEM_NAME_top_sites.xyz`: This `xyz` file contains all the top sites across your system.
2. `SYSTEM_NAME_bridging_sites.xyz`: This `xyz` file contains all the bridging sites across your system.

3. `SYSTEM_NAME_three_fold_sites.xyz`: This `xyz` file contains all the three-fold sites across your system.
4. `SYSTEM_NAME_four_fold_sites.xyz`: This `xyz` file contains all the four-fold sites across your system.

In these systems, the **binding sites are represented as hydrogen atoms**. Examples of `SYSTEM_NAME_top_sites.xyz`, `SYSTEM_NAME_bridging_sites.xyz`, and `SYSTEM_NAME_three_fold_sites.xyz` of a Cu_{78} cluster are given below. This particular example does not contain any four-fold sites.

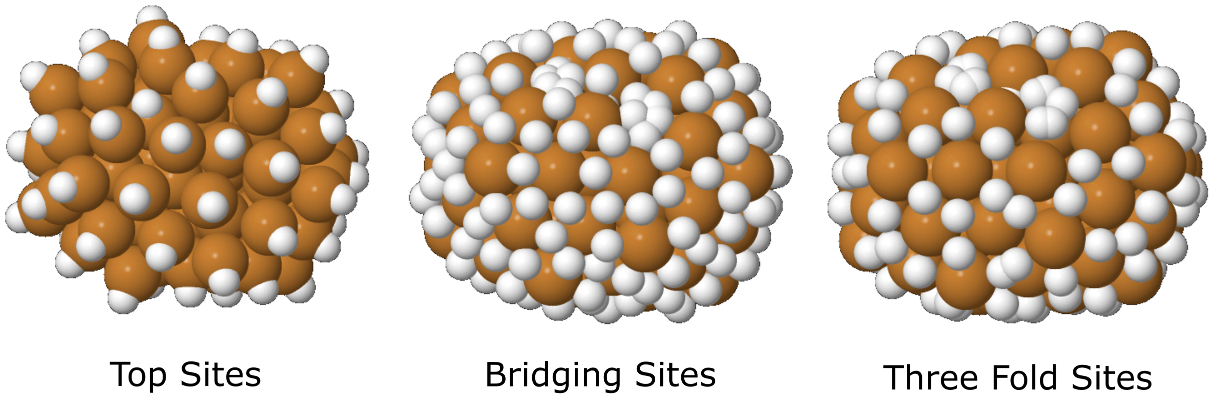


Fig. 11: Top sites (`SYSTEM_NAME_top_sites.xyz`), bridging sites (`SYSTEM_NAME_bridging_sites.xyz`), and three-fold sites (`SYSTEM_NAME_three_fold_sites.xyz`) across this cluster, where each of these sites are represented with hydrogen atoms.

The `xyz` files that are found in `Part_B_All_Systems_with_Adsorbed_Species\ADSORBATE\ADSORPTIONSITE` are labelled as `ADSORBATE_ADSORPTIONSITE_Label_Index.xyz`, where:

- `ADSORBATE`: The adsorbate you want to adsorb to the surface of your system.
- `ADSORPTIONSITE`: The type of surface site that the adsorbate is bound to.
- `Label`: The label of the binding site.
- `Index`: The index of the binding site.

For example, if you want to see the COOH molecule bound to three-fold site labelled 44, you would go to `Part_B_All_Systems_with_Adsorbed_Species > COOH > Three_Fold_Sites` and look at any of the file with `COOH_three_fold_sites_44` in its name. This example is shown below, next to the original three-fold binding site `.xyz` file.

4.7.2 Selecting binding sites using the `Label` command in Jmol

You can view the `Label` of each binding site in Jmol. This is the number that is assign to each of the binding sites. To do this, first open the xyz file in the terminal:

```
jmol SYSTEM_NAME_top_sites.xyz
jmol SYSTEM_NAME_bridging_sites.xyz
jmol SYSTEM_NAME_three_fold_sites.xyz
jmol SYSTEM_NAME_four_fold_sites.xyz
```

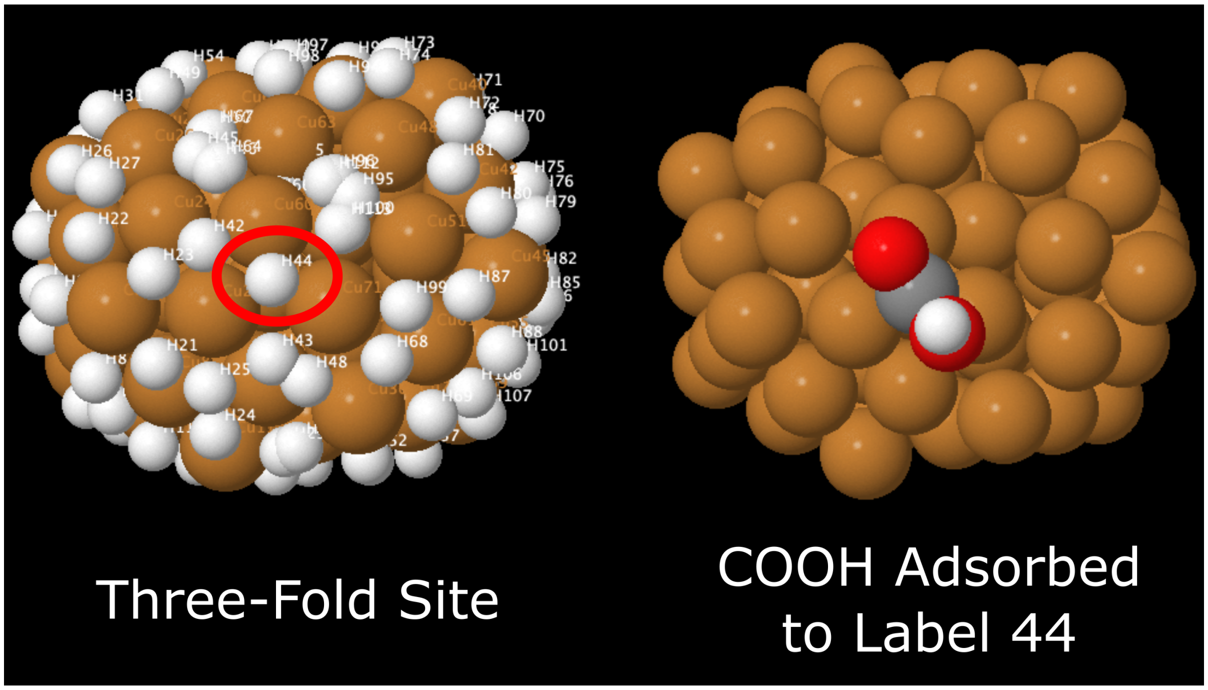


Fig. 12: This example cluster with a COOH molecule adsorbed to three-fold site labelled 44 (just one of the orientations is shown in this example).

This will open up your cluster/surface model in Jmol. Then in the Jmol menu click `Display > Label > Name`. This will label all the atoms by their element symbol and `Label`, where the binding site are labelled `HX`, where `X` is the `Label` of the hydrogen/binding site in the cluster/surface model.

4.7.3 Advice on how I Choose `xyz` Files for VASP Optimise with `Adsorber`

The way that I have found the best use of these four `xyz` files is by colouring in the hydrogens in Jmol that I want to bind all adsorbate to on this system. This can be the same colour, or by colour in the different types of sites in different colours that are of use to use. For example, in the following figure I have coloured the binding sites of interest across this Cu_{78} cluster green for icosahedral sites, interesting sites about the middle of the cluster in yellow, other interesting corner sites in blue, and vacant five-fold vertex sites in red.

Once you have coloured in your atoms of interest, you can obtain the indices of binding sites of interest by saving your Jmol system as a state file. You can do this by clicking on the notepad icon circled in red in the figure below:

If you open this file in a notepad program (for example in Sublime, see https://www.sublimetext.com/) and scroll down to the section called `function _setModelState()`, the indices of the atoms your have coloured are given here. For example in the section of the state file shown below:

```
function _setModelState() {

  select ({89 96 98 102 115 121 131 135});
  color atoms opaque [xffff00];
  select ({0:77});
  Spacefill 1.2;
  select ({104 126});
  color atoms opaque [xff0000];
  select ({92 93 132});
```

(continues on next page)

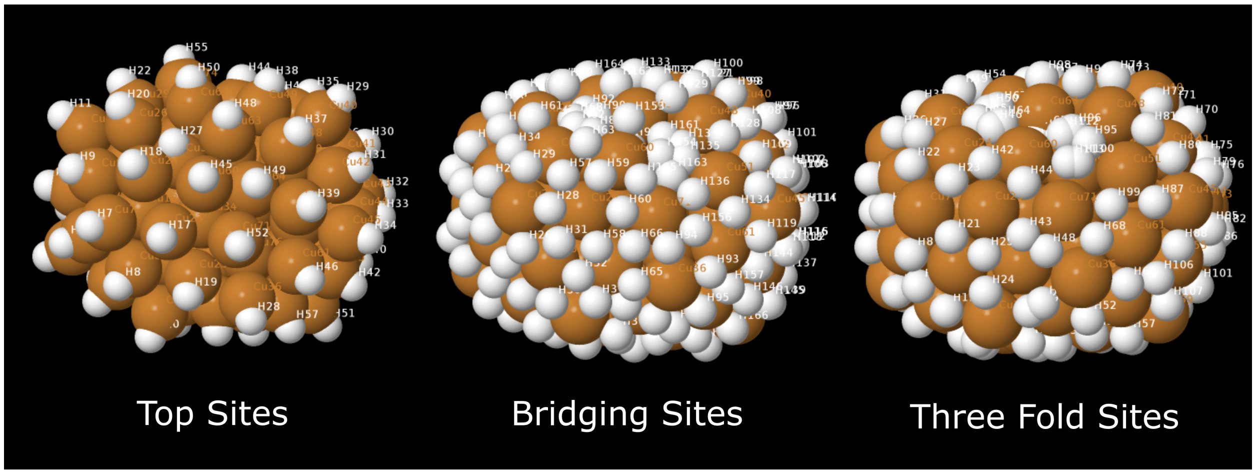


Fig. 13: Top sites (`SYSTEM_NAME_top_sites.xyz`), bridging sites (`SYSTEM_NAME_bridging_sites.xyz`), and three-fold sites (`SYSTEM_NAME_three_fold_sites.xyz`) across this cluster, where each of these sites are represented with hydrogen atoms. Each site is labelled `HX`, where `X` is the `Label` for that binding site.

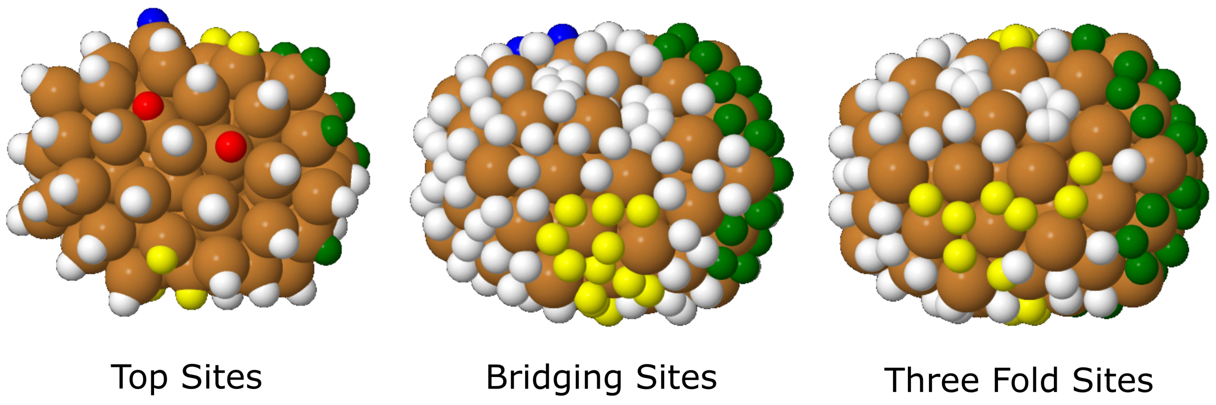
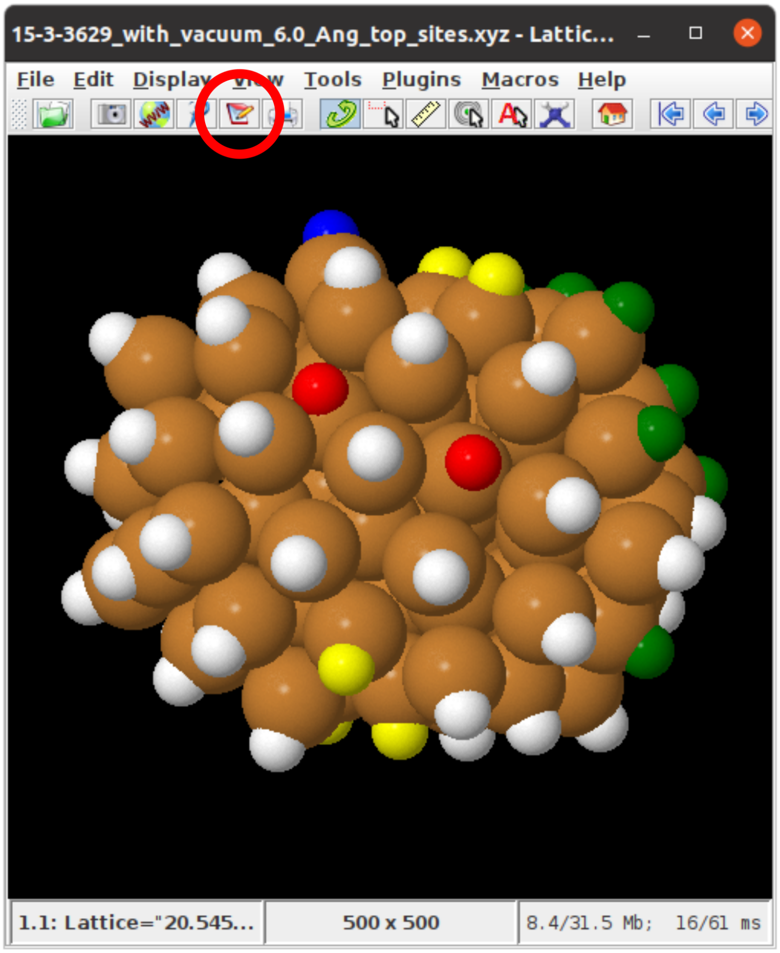


Fig. 14: Top sites (`SYSTEM_NAME_top_sites.xyz`), bridging sites (`SYSTEM_NAME_bridging_sites.xyz`), and three-fold sites (`SYSTEM_NAME_three_fold_sites.xyz`) across this cluster, where each of these sites are represented with hydrogen atoms. Colours are used to help record which binding sites have been noted of interest for further optimisation with VASP.


15-3-3629_with_vacuum_6.0_Ang_top_sites.xyz - Lattic...
File Edit Display View Tools Plugins Macros Help
1.1: Lattice="20.545...
500 x 500
8.4/31.5 Mb; 16/61 ms

(continued from previous page)

```
  color atoms opaque [x0000ff];
  select ({78:135});
  Spacefill 0.66;
  select ({106:109 112 113 118 119 124 130});
  color atoms opaque [x008000];

  hover "%U";

  frank off;
  font frank 16.0 SansSerif Plain;
  select *;
  set fontScaling false;

}
```

Every `select ({Indices});` line that comes before a `color atoms` line are the indices of the atoms that you have select for binding adsorbates to. You can copy the `ADSORBATE_ADSORPTIONSITE_Label_Index.xyz` files from your `Part_B_All_Systems_with_Adsorbed_Species\ADSORBATE\ADSORPTIONSITE` folder to the corresponding `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files\ADSORBATE\ADSORPTIONSITE` folder by hand. Here, you want to look at the `Index` of your `ADSORBATE_ADSORPTIONSITE_Label_Index.xyz` and compare these to your entries in the relevant `select ({Indices});` lines.

4.7.4 How to automate the copying of these `xyz` files: Using `copy_files_from_folder_B_to_C.py`

The process of choosing which binding sites to use for adsorbating adsorbates to can be a laborious process. For this reason, I have created another python script called `copy_files_from_folder_B_to_C.py` which can copy the relevant files for you. An example of this is shown below:

```
from Adsorber import Copy_Files_from_Folder_B_to_Folder_C

adsorbates = ['CO', 'COOH']

top_sites = {'Weird_Sites_Yellow': '89 96 98 102 115 121 131 135', '5_Fold_Vertex_
↪Site_Red':'104 126', 'Weird_Corners_Blue':'92 93 132', 'Ico_Sites_Green':'106:109␣
↪112 113 118 119 124 130'}
bridge_sites = {'Weird_Sites_Yellow':'100 109 114 115 119:122 132 135 141:143 148 149␣
↪160 171', 'Other_5_fold_Sites_Blue':'99 123 126 127 130 131 150 152 157 158 227 229␣
↪241 245', 'Ico_Like_Green':'155 164 173:188 191 193 195 197:204 214 217:223 225 228␣
↪235 242 244'}
three_fold_sites = {'Weird_Sites_Yellow':'93 95 97 98 101 102 106 109:111 114:117 120␣
↪125 136 137 145 174:176 187', 'Ico_Like_Green':'132 133 138:140 147:163 165:170␣
↪178:186 191'}
four_fold_sites = {}

Copy_Files_from_Folder_B_to_Folder_C(adsorbates, top_sites, bridge_sites, three_fold_
↪sites, four_fold_sites)
```

This program will copy of the relevant `ADSORBATE_ADSORPTIONSITE_Label_Index.xyz` files from your `Part_B_All_Systems_with_Adsorbed_Species` folders to your `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files` folder. All orientations/rotations of adsorbates are included, therefore you will need to delete those orientations/rotations you do not want to include. These will be place in folders based on the names you gave the binding sites in the dictionaries. For example, you will find the folders `Weird_Sites_Yellow`, `5_Fold_Vertex_Site_Red`, `Weird_Corners_Blue`, `Ico_Sites_Green` in your `Top_Sites` folder in `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files`.

4.7.5 What To Do Once You Have Placed Selected `xyz` Files Into `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_file`

Once you have placed the selected adsorbate+system `xyz` files into `Part_C_Selected_Systems_with_Adsorbed_Species_` of the desired orientations/rotations, you can proceed to Part C (*Part C.1: Preparing Selected Adsorbed Systems For VASP Optimisation*).

4.8 Part C.1: Preparing Selected Adsorbed Systems For VASP Optimisation

After you have selected the binding sites to adsorb adsorbates onto and have placed their associated `xyz` files into `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files` (with the desired orientations/rotations), we can proceed to preparing the VASP files for these systems with adsorbates. To do this, **set the** `Step_to_Perform` **variable in the** `Run_Adsorber.py` **script to** `'Part C'`:

```
Step_to_Perform = 'Part C'
```

As mentioned previously in *Part A: How to optimise your system initially*, You also need to create a folder called `VASP_Files` that contains the following files and folders:

- `INCAR`: This is a VASP file that contains all the information required to run the VASP job (https://www.vasp.at/wiki/index.php/INCAR and https://cms.mpi.univie.ac.at/vasp/vasp/INCAR_File.html).
- `KPOINTS`: The KPOINTS file is used to specify the Bloch vectors (k-points) that will be used to sample the Brillouin zone in your calculation (https://www.vasp.at/wiki/index.php/KPOINTS).
- `POTCARs`: This is a folder that contains all of the `POTCAR` files for all of the different elements in your models. Each of the `POTCAR` files in this folder need to be labelled as `POTCAR_XX`, where `XX` is the symbol for the particular element. For example, for the POTCAR to describe Cu, you want to name the POTCAR as `POTCAR_Cu`, the POTCAR for C should be called `POTCAR_C`, the POTCAR for H should be called `POTCAR_H`, ...

You want to also include any other files that will be needed. For example, if you are running VASP with the BEEF functional, you need to include the `vdw_kernel.bindat` file in the `VASP_Files` folder. An example of `VASP_Files` folders can be found in Adsorber Examples on Github[17].

You also want to make sure that your `Run_Adsorber.py` script also includes the `slurm_information` dictionary that contains the information required to make the `submit.sl` file. The `submit.sl` file is used to submit a VASP job to Slurm. The information required in the `slurm_information` dictionary can be found at *Information required to make submit.sl siles for submitting files to Slurm*.

Once you have done all of these requirements, you can then run the `Run_Adsorber.py` script in the terminal:

```
python Run_Adsorber.py
```

[17] https://github.com/GardenGroupUO/Adsorber/tree/main/Example

4.8.1 What will `Run_Adsorber.py`: Part C do?

`Run_Adsorber.py` will take all the `.xyz` that you have placed in `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files` and convert them into files ready to be run in VASP with the Slurm Workload Manager.

`Adsorber` will create a new folder called `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP` that contain VASP folders of your selected systems with adsorbates. Each of these VASP folders contain a `POSCAR` of the system with adsorbate, as well as the `INCAR`, `KPOINTS`, `POTCAR`, and `submit.sl` files, as well as any other files that you need for your VASP calcuations.

If the VASP folder exists and it contains a `POSCAR`, this `POSCAR` will not be replaced as you may have updated the `POSCAR` if your VASP job entered prematurally without converging. If you do want to force override all `POSCAR` files, you will want to set `part_c_force_create_original_POSCAR = True` in your `Run_Adsorber.py` script.

Note: This `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP` folder may get big, so just check the amount of space that the newly created `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP` is taking up as it is being created.

4.8.2 I accidentally gave wrong settings in the `INCAR` or `submit.sl` files, or something about my `KPOINTS` or `POTCAR`. What should I do?

If you realise you have entered in wrong settings in the `INCAR` or `submit.sl` files, or your `KPOINTS` or `POTCAR` files are wrong, no problem! Make the changes to these files and then rerun your `Run_Adsorber.py` script again. Only those jobs that have not begun to run (i.e. dont have an `OUTCAR`) will have their VASP files `INCAR`, `submit,sl`, `KPOINTS`, `POTCAR`, and other vast files (not the `POSCAR` though) copied over. Adsorber will not touch those jobs that have an `OUTCAR` that are assumed to be running/have finished running.

4.8.3 I want to add more new places that adsorbates can bind to on the surface of the cluster/surface model in Part B, what do I do here?

Run your `Run_Adsorber.py` script once you have included all the new binding sites to your `Part_C_Selected_Systems_with_Adsorbed_Species_to_Convert_into_VASP_files` folder. Running your your `Run_Adsorber.py` script again will add new folders and VASP files of these new arrangements of adsorbates on the surface of your cluster/surface model in your `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP` folder. Your original `POSCAR` will not be changed (unless you have set `part_c_force_create_original_POSCAR = True`. We recommend you not to do this here). Only those jobs that have not begun to run (i.e. dont have an `OUTCAR`) will have their VASP files `INCAR`, `submit,sl`, `KPOINTS`, `POTCAR`, and other vast files (not the `POSCAR` though) copied over. Adsorber will not touch those jobs that have an `OUTCAR` that are assumed to be running/have finished running.

4.8.4 How to submit VASP jobs to Slurm: The `Run_Adsorber_submitSL_slurm.py` program

Once you have run the `Run_Adsorber.py` script with `Step_to_Perform = 'Part C'`, you can submit your jobs. If you use a computer cluster that run the Slurm Workload Manager, you can submit all your jobs at the same time by changing directory into the `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP` folder and running a script called `Run_Adsorber_submitSL_slurm.py`.

```
cd Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP
Run_Adsorber_submitSL_slurm.py
```

Running the `Run_Adsorber_submitSL_slurm.py` script in the terminal will submit all your VASP jobs. The `Run_Adsorber_submitSL_slurm.py` program works by looking through all subdirectories that this program is executed from and looks for folders that contain a `'submit.sl'` file.

- **This program will not submit VASP jobs that are currently running or have been run**. **This program will only submit VASP jobs that do not contain a ``OUTCAR`` file**. Any job that is running or has already run will contain an `OUTCAR` file, which tells `Run_Adsorber_submitSL_slurm.py` that that VASP job is currently running or has already been run.
- `Run_Adsorber_submitSL_slurm.py` will execute all folders that contain a `'submit.sl'` file. However, `Run_Adsorber_submitSL_slurm.py` will not run any `'submit.sl'` files from previously run calculations, which are found in the `Submission_Folder` folders.

If you dont want to run all the jobs in `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP` but just a select few, you want to move into that folder and then type `Run_Adsorber_submitSL_slurm.py` into the terminal. For example, lets say that I only want to run the jobs that are in the directory `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP/COOH_symmetric/Bridge_Sites/Other_5_fold_Sites_Blue`, then we want to move into this directory and then type `Run_Adsorber_submitSL_slurm.py` into the terminal:

```
cd Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP/COOH_symmetric/Bridge_
↪Sites/Other_5_fold_Sites_Blue
Run_Adsorber_submitSL_slurm.py
```

`Run_Adsorber_submitSL_slurm.py` is set up to only allow 1000 jobs to be running or in the queue in slurm. You can change this value in the `Run_Adsorber_submitSL_slurm.py`, however by default slurm usually only allows for 1000 jobs to be running or in the queue at any one time. Before you run `Run_Adsorber_submitSL_slurm.py` you can see how many jobs you are submitting to the queue by running typing `no_of_submitSL_files` into the terminal in the directory you are in. To use this command, you need to include the alias in your `~/.bashrc`:

```
alias no_of_submitSL_files='find . -name "submit.sl" -type f -not -path "*Submission_
↪Folder_*" | wc -l'
```

For example, if I want to find out all the jobs in `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP` I move into this directory and type `no_of_submitSL_files` into the terminal:

```
cd Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP
no_of_submitSL_files
```

If I want to find out all the jobs in `Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP/COOH_symmetric/Bridge_Sites/Other_5_fold_Sites_Blue`, I move into this directory and type `no_of_submitSL_files` into the terminal:

```
cd Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP/COOH_symmetric/Bridge_
↪Sites/Other_5_fold_Sites_Blue
no_of_submitSL_files
```

To find out the number of jobs that are running or are waiting in the queue in slurm, you can type `no_of_jobs_running_or_queued` into the terminal. To use this command, you need to enter this alias into your `~/.bashrc`:

```
alias no_of_jobs_running_or_queued='squeue -u $USER | wc -l'
```

Note: This will give you the number of jobs you have in your slurm queue, plus 1. So whatever number you get from `no_of_jobs_running_or_queued`, minus 1 from it to get the actual number of jobs in your queue. Not suer how to fix this yet.

NOTE: You **CAN** enter more than 1000 jobs into the slurm queue with `Run_Adsorber_submitSL_slurm.py`. If you reach 1000 jobs queued in slurm, `Run_Adsorber_submitSL_slurm.py` will patiently wait for current running jobs to complete and add more of your jobs into the slurm queue as current jobs are completed.

4.8.5 What to do if some jobs need to be resubmit for some reason

If you would like to resubmit one or many jobs for some particualy reason, see *Part C.2: What To Do If Some Jobs Have Not Finished/Converged* for information about the programs for doing this.

4.9 Part C.2: What To Do If Some Jobs Have Not Finished/Converged

In many cases, some of your jobs may have not converged, may not have finished due to an error, or you may want to restart your job with a tigher convergence criteria. In these cases you will want to reset your jobs to be resubmitted. There are way to do this (for example, using the `vfin.pl` and `vef.pl` scripts in the `VTST` toolset, https://theory.cm.utexas.edu/vtsttools/). In Adsorber, we have written a few programs and have set up a few protocols to

1. Find out which jobs have converged or not.
2. To resubmit all jobs or (specific jobs) to slurm.

These programs/protocols are described below.

4.9.1 `Run_Adsorber_determine_unconverged_VASP_jobs.py`: Determine which jobs have converged and which have not

This program is designed to inform you of which VASP jobs have converge and which have not. To run this, move into the folder that you would like to examine all jobs that are within subdirectories of. Then run this program in the terminal. For example, if you want to examine if all VASP jobs from Part A have converged, perform the following in the terminal:

```
cd Part_A_Non_Adsorbed_Files_For_VASP
Run_Adsorber_determine_unconverged_VASP_jobs.py
```

If you want to check if all VASP jobs from Part C have converged, perform the following in the terminal:

```
cd Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP
Run_Adsorber_determine_unconverged_VASP_jobs.py
```

If you want to just want to check if the VASP jobs for a particular adsorbate from Part C have converged, for example if all systems that had CO adsorbed to its surface, perform the following in the terminal:

```
cd Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_VASP/CO
Run_Adsorber_determine_unconverged_VASP_jobs.py
```

This will give you a list of VASP jobs that have converged and have not converged:

```
==============================================
The following VASP jobs CONVERGED
CO_top_sites_53_130 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_53_130)
CO_top_sites_42_119 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_42_119)
CO_top_sites_30_107 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_30_107)
CO_top_sites_32_109 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_32_109)
CO_top_sites_31_108 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_31_108)
CO_top_sites_36_113 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_36_113)
CO_top_sites_47_124 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_47_124)
CO_top_sites_35_112 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_35_112)
CO_top_sites_29_106 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_29_106)
CO_top_sites_41_118 (./CO/Top_Sites/Ico_Sites_Green/CO_top_sites_41_118)
==============================================
The following VASP jobs DID NOT CONVERGE
CO_top_sites_49_126 (./CO/Top_Sites/5_Fold_Vertex_Site_Red/CO_top_sites_49_126)
CO_top_sites_27_104 (./CO/Top_Sites/5_Fold_Vertex_Site_Red/CO_top_sites_27_104)
CO_top_sites_58_135 (./CO/Top_Sites/Weird_Sites_Yellow/CO_top_sites_58_135)
CO_top_sites_12_89 (./CO/Top_Sites/Weird_Sites_Yellow/CO_top_sites_12_89)
CO_top_sites_19_96 (./CO/Top_Sites/Weird_Sites_Yellow/CO_top_sites_19_96)
CO_top_sites_21_98 (./CO/Top_Sites/Weird_Sites_Yellow/CO_top_sites_21_98)
CO_top_sites_44_121 (./CO/Top_Sites/Weird_Sites_Yellow/CO_top_sites_44_121)
==============================================
```

If you just want the names of the jobs and not the directories printed, type `Run_Adsorber_determine_unconverged_VASP_jobs.py False` into the terminal. This will give the following:

```
==============================================
The following VASP jobs CONVERGED
CO_top_sites_53_130
CO_top_sites_42_119
CO_top_sites_30_107
CO_top_sites_32_109
CO_top_sites_31_108
CO_top_sites_36_113
CO_top_sites_47_124
CO_top_sites_35_112
CO_top_sites_29_106
CO_top_sites_41_118
==============================================
The following VASP jobs DID NOT CONVERGE
CO_top_sites_49_126
CO_top_sites_27_104
CO_top_sites_58_135
CO_top_sites_12_89
CO_top_sites_19_96
CO_top_sites_21_98
CO_top_sites_44_121
==============================================
```

4.9.2 What to do if you want to resubmit jobs to slurm

There are two programs that you can use for preparing jobs for resubmission to slurm, depending on what you want to do. The first thing to do is to make any changes to your convergence criteria or other VASP settings. Once you are happy, move on to one of the two pro before that is most suited to what you want to do.

Before preparing jobs for resubmission

Before using either of these programs, you want to first make any changes to the settings that you want to change in your `INCAR` file (and make any corrections that you need to make to your `KPOINTS` and `POTCAR` files if required). For example, if you want to change the geometric convergence criteria you want to change the `EDIFF` tag in your `INCAR` file now.

If you dont need to make any changes to your `INCAR`, do not worry about any of this.

`Run_Adsorber_prepare_unconverged_VASP_jobs.py`: Prepare unconverged VASP jobs for re-submission

If not all your VASP jobs converged, you can setup your VASP calculations to be resubmitted to VASP from the last geometry optimisation step. To do this, you first need to prepare a new python script in the same place on your computer as your `Run_Adsorber.py` called `prepare_unconverged_VASP_jobs.py`. An example of this `prepare_unconverged_VASP_jobs.py` python script is as follows:

```
from Adsorber import Run_Adsorber_prepare_unconverged_VASP_jobs

# A switch that determines what type of resubmnission scheme you would like to perform
prepare_jobs_switch = 'folder' # text

# if you want to resubmit all adsorbate+systems that have an energy above the current␣
↪minimum energy system.
files_with_VASP_calcs = ['Part_C_Selected_Systems_with_Adsorbed_Species_to_Run_in_
↪VASP/COH']
options = {'energies_from_lowest_energy': float('inf')}

# If you want to resubmit certain adsorbate+systems given in a text file.
path_to_resubmission_list_file = 'Part_D_Results_Folder/Similar_Systems_CHO.txt' #␣
↪example of path_to_resubmission_list_file as a string for a single file
# path_to_resubmission_list_file = ['Part_D_Results_Folder/Similar_Systems_CHO.txt',
↪'Part_D_Results_Folder/Similar_Systems_COOH.txt', 'Part_D_Results_Folder/Similar_
↪Systems_CO.txt'] # example of path_to_resubmission_list_file as a list of files.

# Information required to prepare jobs with selected switch
main_information = {'files_with_VASP_calcs': files_with_VASP_calcs, 'options':␣
↪options}
#main_information = {'path_to_resubmission_list_file': path_to_resubmission_list_file}

# if you would like to prepare jobs even if they have already converged, change this␣
↪to True
force_prepare = false
# If you want to also update the VASP files while performing this task
update_VASP_files = False

slurm_information = {}
slurm_information['project'] = 'uoo02568'
slurm_information['partition'] = 'large'
```

(continues on next page)

(continued from previous page)

```
slurm_information['time'] = '72:00:00'
slurm_information['nodes'] = 1
slurm_information['ntasks_per_node'] = 12
slurm_information['mem-per-cpu'] = '1200MB'
slurm_information['email'] = 'yourslurmnotificationemailaddress@gmail.com'
slurm_information['vasp_version'] = 'VASP/5.3.5-intel-2017a-VTST-BEEF'
slurm_information['vasp_execution'] = 'vasp_cd'

Run_Adsorber_prepare_unconverged_VASP_jobs(prepare_jobs_switch,main_information=main_
↪information,slurm_information=slurm_information,force_prepare=force_prepare,update_
↪VASP_files=update_VASP_files)
```

There are five variables to specify in this script. These are :

- `prepare_jobs_switch` (*str.*): This switch indicates how this program will prepare your jobs. There are two options for this switch:
  - `'folder'`: This program will go through selected folders that include all the jobs you would like to prepare.
  - `'text'`: This program will prepare only those jobs that have been included in a given text file.
- `main_information` (*dict.*): This dictionary holds the information required to run this program with `prepare_jobs_switch = 'folder'` or `prepare_jobs_switch = 'text'`. These are:
  - For `prepare_jobs_switch = 'folder'`:
    * `files_with_VASP_calcs` (*list*): This is the list of directories that contains the jobs you would like to resume. This program will look through the directories in this list as well as all the subdirectories in this list and will resume all the jobs within these directories and subdirectories.
    * `energies_from_lowest_energy` (*float*, optional): This variable allows the user to only prepare those jobs that within `energies_from_lowest_energy` eV of the lowest energy adsorbate+system. Any adsorbate+systems that are above `energies_from_lowest_energy` eV of the lowest energy adsorbate+system will not be prepared for resuming. Default: `energies_from_lowest_energy = float('inf')` (Figure this out, maybe remove)
  - For `prepare_jobs_switch = 'text'`:
    * `path_to_resubmission_list_file` (*str./list/tuple*): This is the path to the text file(s) that contains all the paths the jobs that you want to resume. This can be given as a string to point to a single text file, or as a list that points to many text files. See the above code for an example of `path_to_resubmission_list_file'' as a list`. NOTE: You can make this list using the `Run_Adsorber_determine_unconverged_VASP_jobs.py` program; see *Run_Adsorber_determine_unconverged_VASP_jobs.py: Determine which jobs have converged and which have not* for more information.
- `force_prepare` (*bool.*): This setting will only prepare those jobs that have not converged. If you set this to `True`, this program will prepare all files in dictories and subdirectories if they are converged and not converged. Default: `False`.
- `update_VASP_files` (*bool.*): If this variable is set to `True`, the files that are in your `VASP_files` folder will be copied into the job that are prepared. This allows you to make changes to the files in your `VASP_files` folder that you would like to adopt in the jobs you prepare, such as changing the convergence criteria in the `INCAR`. If you set this to `False`, the original VASP files from the Job will be used. Default: `False`.
- `slurm_information` (*dict.*): This dictionary contains all the information required to create the `submit.sl` scripts. See *5) Information required to make submit.sl siles for submitting files to Slurm* for more information about the settings to place in this dictionary.

What will `Run_Adsorber_prepare_unconverged_VASP_jobs.py` do?

For each job that is setup for resubmission, the `CONTCAR`, `INCAR`, `KPOINT`, `OUTCAR`, `POSCAR`, and `submit.sl` files , as well as any output and error files created by slurm during the VASP optimisation, are moved to a folder called `Submission_Folder`. The `CHG`, `CHGCAR`, `DOSCAR`, `EIGENVAL`, `IBZKPT`, `OSZICAR`, `PCDAT`, `PCDAT`, `REPORT`, `vasprun.xml`, `WAVECAR`, `XDATCAR` files are deleted, the last image written in the `OUTCAR` is used as the new `POSCAR`, and the old `OUTCAR` is deleted. `Run_Adsorber_prepare_unconverged_VASP_jobs.py` **will also prepare any VASP jobs for resubmission that had issues, because the `OUTCAR` or `CONTCAR` could not be loaded.** In this case, the POSCAR used will be the original POSCAR. Files from the previous VASP job run will be stored in a folder called `Submission_Folder` with `Issue` included in the label.

What to do if you have run `Run_Adsorber_prepare_unconverged_VASP_jobs.py`, but you then want to change the VASP files or the `submit.sl` script before resubmitting jobs to slurm

If you have already run the `Run_Adsorber_prepare_unconverged_VASP_jobs.py` but decide you want to change some of your VASP files, such as using a different convergence criteria or change other parameters in `INCAR`, `KPOINT`, or other files, you can do this by rerun your `Run_Adsorber.py` script again. To do this:

1. Make the necessary changes to your `INCAR`, `KPOINT`, or other files in your `VASP_Files` folder.
2. Make the necessary changes to your `submit.sl` script by making changes to your `slurm_information` dictionary in your `Run_Adsorber.py` script.
3. Make sure that the `part_to_perform` variable in your `Run_Adsorber.py` script is set to `'Part C'` (`part_to_perform = 'Part C'`).
4. Run your `Run_Adsorber.py` script in the terminal:

```
python Run_Adsorber.py
```

=> If you want to change the convergence criteria before you resubmit your unconverged VASP jobs, perform the steps as above, making sure you change the `EDIFFG` tag in the `INCAR` file suppied in the `VASP_Files` folder. For example, if you want to tighten your convergence criteria, change your value of `EDIFFG` in your `INCAR` file so it is closer to 0.0 eV or 0.0 eV/Ang.

What to do when you are ready to resubmit VASP jobs to slurm

When you are ready to resubmit these jobs, see *How to submit VASP jobs to Slurm: The Run_Adsorber_submitSL_slurm.py program* for information about the `Run_Adsorber_submitSL_slurm.py`, a program for automatically resubmitting jobs to slurm.

4.10 Part D: Gathering Information from VASP Calculations

At any point during and after your VASP calculations have been running (during Part A and Part C), you can run a python program that will gather information about your VASP calculations, such as the energies of your systems with attached adsorbates, and if your VASP calculations have converged or not. You can run this program by typing `Run_Adsorber_Part_D_gather_information.py` into the terminal in the same folder that you ran `Run_Adosrber.py` from:

```
Run_Adsorber_Part_D_gather_information.py
```

This will create an excel file called `Part_D_Information_on_VASP_Calculations.xlsx` that contains various information about your VASP calculations. All adsorbates will be collected together based on their name before any _. For example, if you tried adsorbing `COOH` in two ways, called `COOH_symmetric` and `COOH_O_tilted`, `Run_Adsorber_Part_D_gather_information.py` will group the information from these two sets of calculations together because they both start with `COOH` before the _, telling `Run_Adsorber_Part_D_gather_information.py` they both involve adsorbing `COOH` to the surface of your model.

If you only want to get information for a few different adsorbates, include these adsorbates after typing `Run_Adsorber_Part_D_gather_information.py` into the terminal. For example, if we only want to gather information on the adsorbates `CHO` and `COOH`, we do the following:

```
Run_Adsorber_Part_D_gather_information.py CHO COOH
```

The following information is included in the `Part_D_Information_on_VASP_Calculations.xlsx` excel spreadsheet:

- `'Job'`: The Job ID of the job assign by slurm.
- `'Project'`: The name of the project.
- `'Job Name'`: The name of the job, as given by Adsorber.
- `'Path'`: The path from the folder that your `Run_Adsorber.py` script was run from to the VASP job.
- `'Description'`: A description of the job.
- `'Time submitted for'`: The amount of time that the user submitted the job for.
- `'Date Submitted'`: The date when the job began.
- `'Date Finished'`: The date when the job ended.
- `'Time Elapsed (hrs)'`: The amount of time that actually elapsed (will be shorter or the same time as `'Time submitted for'`).
- `'Max. Memory (Gb)'`: The maximum amount of memory that was used by VASP for this job.
- `'Energy (eV)'`: The energy of the system, adsorbate, or system+adsorbate.
- `'Rel. Energy (eV)'`: The energy of the system relative to the lowest energy system with that adsorbate (not included in the `Originals` tab).
- `'Converged?'`: Will indicate if the VASP job converged or not.
- `'Similar to'`: This will indicate if there are any other jobs that finished where the adsorbate moved into similar positions. This is not complete, so expect for some VASP jobs that finished with adsorbates in the same place to not be given here.
- `'No of surface atoms adsorbed to'`: This will give the number of atoms thats that the adsorbate is bound to, as well as other information about which atoms in the adsorbate are bound to which surface atoms of your surface/cluster. This is given as `[adsorbate atom->surface atoms]`.
- `'Notes'`: An empty cell for you to write any notes in.

While this excel spreadsheet will tell you if a job converges or not, it doesn't tell you if VASP has done something stupid, unexpected, or unintended. You will want to go though each of your VASP calculations and check to make sure you are happy with those VASP calculations or not. **This excel spreadsheet is intended to assist your analysis, not to replace your analysis**.

4.10.1 Part D: Supplementary Methods for tightening convergence criteria and resuming VASP jobs

Often because there are lots of ways that adsorbates can be bound to a cluster, many VASP jobs are required where an adsorbate is bound to various surface sites about a surface or cluster. This means that 200 to 2000+ VASP jobs need to be completed to understand the preferable binding site for an adsorbate upon a surface or cluster. For this reason, I often will perform `Adsorber` at a low convergence (for example 0.03 eV) and then once I know which sites are most preferable for an adsorbate to bind to, only perform a tigher convergence (0.01 eV) on those lowest energy sites.

Furthermore, in some cases, some of your VASP jobs will converge the adsorbate to the same place on the surface as many other VASP jobs (for example, adsorb a COOH to the same top-top site starting at different positions on your surface). You can use the `Part_D_Information_on_VASP_Calculations.xlsx` excel spreadsheet to look to see this, by looking at the columns `Similar to` and `No of surface atoms adsorbed to`. If either of these are the same as other systems (other rows), this may indicate that those systems (rows) are equivalent.

The following methods can help you figure out if different VASP jobs have converged to the same place and help resume any jobs that you want to tighten the convergence of

`Run_Adsorber_compare_systems_with_same_binding.py`: Comparing systems where adsorbate binds to the same surface atoms

As well as using the `Similar to` and `No of surface atoms adsorbed to` columns of the `Part_D_Information_on_VASP_Calculations.xlsx` excel spreadsheet, you can also use this program to help figure out which systems converged to the same place. This program will look through your `Part_D_Information_on_VASP_Calculations.xlsx` excel spreadsheet, compare those systems that have the same `No of surface atoms adsorbed to`, and write files to help you analyse those systems that may be similar.

It is useful to know which system converge to the same place, because if you want to tighten the convergence criteria you don't need to waste computer time resuming calculations for system that have already converged to the same place with looser convergence criteria. Tightening the convergence of these system may run in identical fashions, therefore only wasting computer time.

To use this program, you want to first move into the same directory as your `Run_Adsorber.py` file and then type `Run_Adsorber_compare_systems_with_same_binding.py` into the terminal:

```
cd into_the_same_directory_as_your_Run_Adsorber_script
Run_Adsorber_compare_systems_with_same_binding.py write_similarity_traj_files upper_
↪energy_limit
```

You can have the following optional inputs:

- `write_similarity_traj_files` (*bool*): If true, this program will write all the systems that were identical/similar to the same `.traj` file. This is to allow the user to check that all these systems are infact the same or similar enough to be regarded the same ('Default: `false`').
- `upper_energy_limit` (*float*): By default, all of the lowest energy version of identical final state systems is written to this system. If you only want to write those that are X.XX eV above the minimum energy system, set this value to X.XX eV. For example, if you only want to record those systems that are 0.5 eV above the minimum energy system, set this value to 0.5. (Default: record all lowenergy energy versions of identical states).

This will create a folder called `Similar_Systems` into your `Part_D_Results_Folder` directory. This folder will contain subdirectories that are name in the same way as in your `Part_D_Information_on_VASP_Calculations.xlsx` excel spreadsheet (for example, `1 (C1 [79->25]),1 (O1 [80->66])`. However this will be relabelled so that spaces are changed to `_`, `[`, `]`, `(`, and `)` removed, `,` to `+` and `->` to `to`, e.g. `1 (C1 [79->25]),1 (O1 [80->66])` goes to `1_C1_79to25__1_O1_80to66`). These alternative names are also give in the

`Part_D_Information_on_VASP_Calculations.xlsx` excel spreadsheet, in the cell to the right of this cell. In these folders will contain:

- `similar_systems.txt`: This file contains all the job_names, energies, and paths of VASP jobs that may have converged to the same place
- a `traj` file: This contains all the final states of your jobs. The images in this `traj` are ordered in the same manor as given in `similar_systems.txt`. The images in this `traj` may all look the same, because these jobs may have converged to the same place.
- `xyz` files: These are xyz files of the final states that jobs had reached before finishing. These `xyz` files may all look the same, because these jobs may have converged to the same place.

`Run_Adsorber_prepare_unconverged_VASP_jobs.py`: Prepare Jobs for resubmission

You may want to rerun some of your jobs, either because they finished with errors, did not converge, or because you want to change the convergence criteria for that job (either by tightening or loosening the convergence criteria). See *What to do if you want to resubmit jobs to slurm* for information about how to prepare jobs for being resubmitted to slurm.

`Run_Adsorber_Tidy_Finished_Jobs.py`: Clean up the files for jobs that you are happy with

VASP makes lots of files after it has run. These can be annoying to keep if you are transferring files about. The `Run_Adsorber_Tidy_Finished_Jobs.py` script will get rid of all the unnecessary files that are created from all subdirectories. The files that are removed are: `CHG`, `CHGCAR`, `CONTCAR`, `DOSCAR`, `EIGENVAL`, `fe.dat`, `IBZKPT`, `OSZICAR`, `PCDAT`, `POTCAR`, `REPORT`, `vasprun.xml`, `vaspout.eps`, `WAVECAR`, `XDATCAR`, and `vdw_kernel.bindat`. The `INCAR`, `KPOINTS`, `OUTCAR`, `POSCAR`, and `submit.sl` files are not removed, as well as any output and error files that are created by slurm during the VASP optimsation, are **NOT** removed by this script. To perform this script, move into the folders that can all the subfolders you wish to tidy up and enter `Run_Adsorber_Tidy_Finished_Jobs.py` into the terminal:

```
Run_Adsorber_Tidy_Finished_Jobs.py
```

If you do want to remove all `INCAR`, `KPOINTS`, and `submit.sl` files in these folders as well, move into the folders that can all the subfolders you wish to tidy up and enter `Run_Adsorber_Tidy_Finished_Jobs.py full` into the terminal:

```
Run_Adsorber_Tidy_Finished_Jobs.py full
```

Note: the `Run_Adsorber_Tidy_Finished_Jobs.py` program will not change or remove any files that are in your `VASP_Files` folder.

4.11 Index

4.12 Python Module Index

CHAPTER

FIVE

INDICES AND TABLES

- *Index*
- *Python Module Index*