VASP and Atomic Simulation Environment: Tips and Tricks

Andrew Jark-Wah Wong

June 27, 2024

Abstract

A guide I am making of my favorite scripts that I have used through my Ph.D. to increase my efficiency in VASP5.4 and VASP6 by using the Atomic Simulation Environment. This guide is for users that are semi-experienced with DFT but would like their lives be a bit more easier. Here, i'll explain installation and running vasp through an ASE. There are scripts used to create structures, place addsorbates, and enumerate adsorption sites. While a short (and messy guide), I hope it helps one person on their journey of using VASP. All information will be at my GitHub Repository here

I truly believe I am not the best coder and would very much like input on improving these scripts and this guide. If you have any recommendations or questions regarding this guide, email me at ajwongphd@gmail.com.

Graphical Abstract:

Figure 1: Graphical Abstract. It's just a cat. Not related to anything in this work.

Contents

1 Installing ASE

The Atomic Simulation Environment (ASE) is a Python environment made to assist and perform electronic-structure calculations. Having a decent knowledge of Python is helpful but general coding experience is sufficient as long as you understand variables, loops, and simple things. My explaination of a lot of the intricacies and details regarding ASE will not do it justice. I highly recommend reading the documentation here as it will be your best resource as you learn.

The following steps are required to install Atomic Simulation Environment: (note that these steps are general and applicable based on my experience)

1. Install a SSH terminal to use
2. Set up a Python Environment
3. Install ASE in your Python Environment

Before installing ASE, you will most likely be operating it on a secure shell terminal as you are submitting "jobs" to a queue of a supercomputing cluster (unless you're really running VASP on your local computer...). I recommend MobaXterm as it has X11-forwarding capabilities, nice interface, and penguins (yes penguins). Once you are about login into the terminal and situate yourself, you will need to set up a Python environment. There are several ways to do this but I use miniconda as it's a "compact" version of Anaconda. It includes conda, Python, and a few packages that are only essential. The purpose of this environment is a place to install different Python packages to run Python based codes. ASE is one of these packages.

To set up mini-conda, the following steps need to be done with the following command to execute these steps in the terminal:

1. First, you will need to load Python itself in the terminal. Typing "*module spider python*" will let you know what versions of Python are available. ASE requires Python 3.8 and above (based on June 2024).
   (a) *module load python*
2. Download Mini-conda from their website. The command allows one to download the Python3 64 bit version to the terminal.

(a) `wget https://repo.continuum.io/miniconda/ Miniconda3-latest-Linux-x86_64.sh`

3. Install Mini-Conda once you obtained a .sh file from Step 2. A bunch of lines and questions will come up. Say yes to everything.

   (a) `chmod u+x Miniconda3-latest-Linux-x86_64.sh ./Miniconda3-latest-Linux-x86_64.sh`

4. Once you have said yes to everything, name and create your conda environment.

   (a) `conda create --name "give\_what\_ever\_name\_you\_want" python=3.6.8`

5. Once installed, you can activate the environment. You will need to do this every time you log in if you do not have the "source activate {environment}" command in your .bashrc file.

   (a) `chmod u+x Miniconda3-latest-Linux-x86_64.sh ./Miniconda3-latest-Linux-x86_64.sh`

6. Once installed, you can activate the environment. You will need to do this every time you log in if you do not have the "source activate {environment}" command in your .bashrc file.

   (a) `chmod u+x Miniconda3-latest-Linux-x86_64.sh ./Miniconda3-latest-Linux-x86_64.sh`

   (b) The environment is activated when "base" becomes "your environment name"

   (c) For example: `(base) meowman123@submit: >` will be `(your environment name) meowman123@submit: >`

   (d) You will need x11 forwarding activated. MobaXterm has this intrinsicly in its terminal. Others you will need things like "Xming" and such.

   (e) Once you see your environment has been activated, you can now run Python scripts and packages in the terminal.

Now that you have set up your Python environment, you can install ASE. The simplest way once you have your environment open is the following command:

pip install ase

There will be other packages needed but this is everything you need to now run ASE! Feel free to look up available python packages to install. "pip install" is the brute force way to install many packages that I found has not failed me... yet..

2 Basics of ASE

You have survived the most mudane part. A few commands you probably want to be aware of.

The first is to utilize the graphical user interface (gui) to view POSCAR, CONTCAR, and OUTCAR. The following command allows you to visualize these files within the terminal as the coordinates are treated as ASE objects.

ase gui POSCAR

Figure 2 shows what the interface looks like. [Insert Figure of Interface] A few key things you would want to use in this interface are as follow:

1. Saving current POSCAR/CONTCAR that are in direct coordinates as new POSCAR, CONTCAR, etc to convert them to cartesian coordinates.

2. Rotate and move atoms. (Note: ctrl+M does not work in mobaxterm for some reason for the latter).

3. Add atoms and molecules based on coordinates relative to a highlighted atom.

4. If an OUTCAR is visualize, you can visualize the SCF cycle and how the energy, forces, geometry, and more changes as VASP optimizes your structure.

Of course, more are available if you refer to the ASE documentation but these are my few tips and tricks.

3 Submitting VASP Calculations using ASE

Typical VASP calculations requires the following five components:

1. POSCAR (Positions)
2. POTCAR (Pseudopotenitals of each atom)
3. INCAR (Input File)
4. KPOINTS (Specified K-Point grid for brillioun zone sampling)
5. A submisssion script to submit the job to a supercomputing queue.

Arguably the most painful file to create is the POTCAR,especially if the atom ordering becomes not consistent. ASE frees us from these shackles by simplifying the process of submitting VASP jobs by condensing POTCAR,INCAR, and KPOINTS into one file. The files need to run a VASP calculation **through ASE are as follows:**

1. POSCAR
2. runvasp.py ("ASE Input script")
3. A submission script (vasp)

3.1 ASE Input File

Here, we see that the runvasp.py (can change the name) holds all the information regarding POTCAR, INCAR, and KPOINTS. The same POSCAR is used as you would in a regular VASP calculation. We will go over the components of an input file to run a VASP job through ASE. The runvasp.py file contains three general components to adjust:

1. Define the K-Point Grid
2. Inputs per-usual in an INCAR
3. Specify the exchange correlation functional and specific pseudopotentials if required

First, the K-point grid to sample the Brillouin zone is given in the method of the Monkhorst-Pack. I always recommend proper testing w.r.t the number of irreducible K-points with energy for any new surface slab. The input is similar to VASP but the input is as a tuple. More information regarding the implementation of K-Points in ASE can be found here

The next component is the input. It is stored as a dictionary in python but the inputs are identical to VASP. The syntax is different than a typical VASP input as the inputs are in lowercase and need a comma since it's a dictionary. We provided the "translation" of inputs that ASE accepts here in the runvasp.py script.

Lastly, the greatest of ASE is its ability to not need a POTCAR file created. This is due to the fact that your POSCAR is converted to an ASE object, which ASE can identify the atoms itself! However, two requirements are needed to allow ASE to recognize the needed pseduopotentials:

1. A directory of the pseudopotentials that can be read by ASE.

   These are your typical pseudopotentials read by VASP and put in the environment variable `(VASP_PP_PATH)`.

   These will also need to be specified in the submission script of use.

2. Specify specific pseudopotentials needed outside of the standard.

   If you do not, ASE will just use the standard pseudopotential, which may not be appropriate.

   Refer to the VASP Wiki to determine the correct pseudopotential for your atom(s) of interest

3. Define the exchange correlation functional of use.

These are all the components of the Input ASE script. It may be intimidating at first but it is much more efficient as saves you the pain of creating many POTCARs...

3.2 Submission Script

The submission script is not much different than most other submission scripts. I have provided my submission script "asevasp". The only intricacies I would like to put out before submitting are the following:

1. Ensure the correct path to activate your mini-conda environment is used

   ```
   source activate /PATH TO MINI-CONDA3/
   ```

2. Ensure the correct path to the pseudopotential directory is specified.

   ```
   export VASP_PP_PATH=/blah/blah/vasp_pp/
   ```

Once you take care of these paths (and other specification related to just job submission details), you are ready to submit to a VASP job using ASE. You are already orders of magnitude more efficient by utilizing this framework.

4 POSCAR: Bare Metal Surface Slabs and Adsorbate Placements

The next sections explore how Python scripts using ASE and its packages and construct POSCARs of surface and simple adsorbates. Here, the following current scripts are included currently:

1. `Bare_FCC_Facet.py`: Creates bare fcc surface facets.
2. `addadsorbate_monoatomic_fcc.py`: Places a monoatomic adsorbate on a surface slab.
3. `loop_addadsorbate_monoatomic_fcc.py`: Utilizes a for loop to place a different single adsorbate on different surfaces.
4. `AUTOPLACE.PY`: Enumerate surface sites. (Will need to finish)

We will go over the inputs and outputs of these scripts in the following sections.

4.1 Bare Metal Surface Slabs

ASE is great as the packages allow for concise construction of surface slabs. The `Bare_FCC_Facet.py` is the python script of interest. The purpose of this script is to create a POSCAR of a bare fcc metal surface facet using ASE. The ase.build package imports key modules (ex: fcc111,fcc110, and fcc100) is built to know the lattice constants of these different surface and properly obtain the correct lattice size and shape. This I usually run these scripts in the terminal and directory of interest. Here are the few inputs needed to create the surface slab as its associated variable:

1. Define the metal of interest (variable = metal)
2. Define the surface facet (variable = `facet_int`)
3. Define the Slab Specifications

   The size consist of (x,y,L), where x and y defines a x by y surface slab and L is the number of layers.

The amount of vacuum applied. Note: It's the amount to apply to each side. (If 7.5 Åis denoted, the total vacuum region is 15 Å).

That is all the inputs needed! It seems to easy but that's what is so great about ASE. To run the script (assuming it's in the terminal and directory of interest), simply run the following command:

python INSERT.py

Make sure your conda environment is active during this. This command generally runs any .py script using Python. The script will create the surface slab as specified and constrain the layers. Double check the layer constraint (variable = z) as I set these as constants for freezing the bottom three layers of a five layer slab. Then, the script creates a ase object (.vasp) file and converts it to a POSCAR, which is places in a directory based on the specifications. For example, if a Cu(111) surface was built, a directory called `Cu_fcc111_bare.vasp` is made with the POSCAR. This distinguishes POSCARs created. Lastly, a gui will show the created POSCAR. That is it and a quick way to create slabs of different sizes. This definitely beats cutting the bulk slab and such in AMS or something (if you know, you know...).

4.2 Adsorption of monoatomic adsorbate

Now, we can apply similar concepts to create a POSCAR of a surface slab with a monoatomic adsorbate.

The script of interest is `addadsorbate_monoatomic_fcc.py`. This script utilizes same principles as the `Bare_FCC_Facet.py`, where a bare metal surface facet is built in a similar fashion. However, we now have to specify the monoatomic adsorbate of interest (more complex adsorbates are considered later) and it's adsorption site. A summary of the inputs needed are denoted in the following:

1. Define the metal of interest (variable = metal)
2. Define the surface facet (variable = `facet_int`)
3. Define the Slab Specifications

   The size consist of (x,y,L), where x and y defines a x by y surface slab and L is the number of layers.

The amount of vacuum applied. Note: It's the amount to apply to each side. (If 7.5 Åis denoted, the total vacuum region is 15 Å).

4. **Specify the monoatomic Adsorbate (variable = adsorbate)**
5. **Specify the adsorption site (variable = position)**
6. **Specify the height of the adsorbate relative to surface (variable = height)**

As you can see, the first three inputs are the same as before (since I copied pasted it here). However, we now have to specify the adsorbate, the adsorption site, and the height. The adsorption site of interest is different depending on the facet. I have denoted the options in the script. The script essentially takes the bare surface slab (slab) and combines the adsorbate onto the slab using the addadsorbate function. This creates a new slab called adslab, which is the slab with the adsorbate. Then, again, a POSCAR is made and a unique directory is made with the POSCAR, which is called, `Metal_FACET_Adsorbate_AdsorptionSite.vasp`.

The applications of this approach is an easy way to enumerate different preferential adsorption sites for a monoatomic adsorbate. Additionally, you can perform a coverage test by implementing a for loop for the surface slab size. For example, POSCARs of a 4x4, 3x3, 2x3, and 2x2 slab of H* adsorption can easily generated to study coverage effects. Here are some ideas and the potential of ASE as these ideas are a bit painful to do by hand... If For loops excite you, the next section is for you.

4.2.1 Utilizing For Loops to create many POSCARs:Our Best Friend

Here, the beauty and power of ASE and Python shows here.

Here in `loop_addadsorbate_monoatomic_fcc.py`, it is a modification of the previous code. It's actually the same code but here I show an example where we loop through different metals, adsorption sites, and adsorbates. Here, the script utilizes two for loops, one to iterate through the list of metals and the other for the adsorbates. I will then get 4 POSCARs since I have two metals specified and two adsorbates described in one click. You can iterate and make the list longer for different variables. You can simultaneously iterate through different surface facets, adsorption sites, slab sizes, and more.

The caveat is **these variables must now be in a form of a list rather than a float**

Hopefully up to know, you can see how much time can be saved by learning a bit from Python. I am definitely not a great coder but these scripts have let me set up VASP calculations in max 10 percent of the usual time, especially with material screening and such.

Next, we will discuss non-fcc surfaces, more complex adsorbates, and a tool to enumerate all adsorption sites.

5 Bash Scripts

Stuff about my bash scripts

6 Conclusions

I conclude.

Acknowledgments

Thanks everyone, especially Jin Li for helping with creating these scripts and expertise!