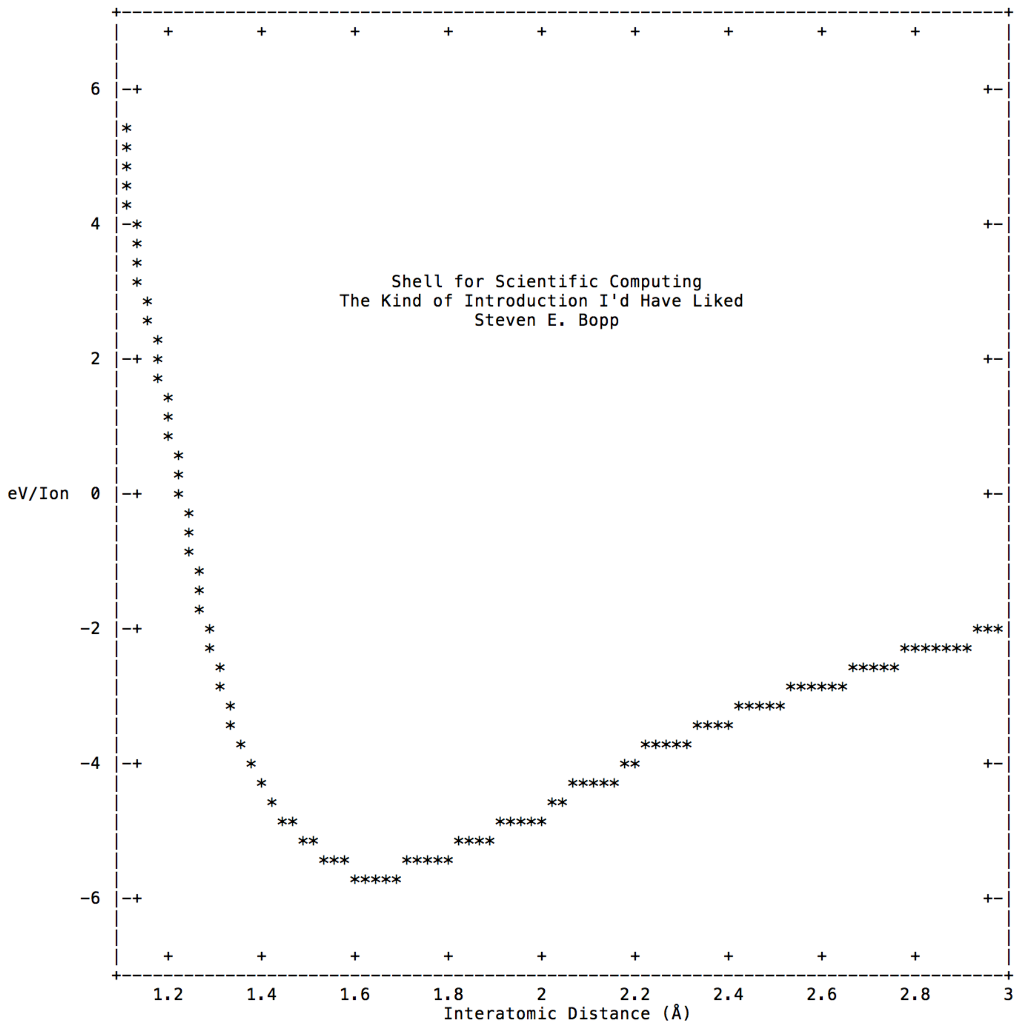


Shell for Scientific Computing
The Kind of Introduction I'd Have Liked

Steven E. Bopp

Shell for Scientific Computing

The Kind of Introduction I'd Have Liked

First Edition. Winter, 2022

Steven E. Bopp, Ph.D.

University of California San Diego

UC eScholarship

ISBN: 979-8-218-11224-0 (electronic; `https://doi.org/10.21221/S2G59Q`)

Publisher: This version was initially released in electronic form on December 15, 2022. Published by eScholarship, University of California. An electronic version of this book is freely available at `https://escholarship.org/uc/item/4qb8927d`.

Cover Art: Graphic produced from an original series of VASP calculations run with automated code developed by Steven Edward Bopp using gnuplot on a wicked Department of Energy supercomputer!

Colophon: This document was typeset using LaTeX; graphics were made using a variety of freely available software including Linux, Inkscape, GIMP, and VESTA.

For Tami, Gregory, and Douglas

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>   <!>   <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
```

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>   <!>   <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
```

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>   <!>   <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
```

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>   <!>   <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
```

Shell for Scientific Computing: The Kind of Introduction I'd Have Liked

Steven E. Bopp, Ph.D.: Materials Science & Engineering

First Edition, Winter 2022

Contents

"There are not many persons who know what wonders are opened to them in the stories and visions of their youth; for when as children we listen and dream, we think but half-formed thoughts, and when as men we try to remember, we are dulled and prosaic with the poison of life. But some of us awake in the night with strange phantasms of enchanted hills and gardens, of fountains that sing in the sun, of golden cliffs overhanging murmuring seas, of plains that stretch down to sleeping cities of bronze and stone, and of shadowy companies of heroes that ride caparisoned white horses along the edges of thick forests; and then we know that we have looked back through the ivory gates into that world of wonder which was ours before we were wise and unhappy."

— H. P. Lovecraft, Celephaïs

1 Introduction

This text is intended to be a concise, examples-driven guide to some ways that the shell may be used for scientific computing, automation of what would be otherwise tedious processes on the computer, collection and navigation of data sets, and automatic conversion and analysis of legacy data that might be otherwise difficult to for modern computers to read. Efforts will be made to add the included scripts into the following Github repository incrementally when time is available to do so: `https://github.com/sbopp/Shell_For_Scientific_Computing_SE_Bopp`. This will be done for ease of use by the reader so that code won't have to be copied and pasted directly from this .pdf document.

Many guides consider very carefully the background and supporting physics in addition to some examples of code and programming; this guide is not like that. This text is more or less purely a guide on how to use computational tools based on some worked examples. Some common errors for different situations are also compiled based on notes from my own research while I was learning how to use these tools. The reader is assumed to be able to understand (with just a bit of figuring on their part) the background theory and physics on which many of the topics discussed herein are based. Citations for more background on specific physics underlying some of the included codes are given at the end in the references section.

That being said, all examples contained within were inspired by scripts and code fragments valuable to my research. I use BASH for automation, navigation, and code execution on remote machines, and for running computations on supercomputers. All of what follows is a compilation (with as many comments within the code and explanations surrounding the code as was reasonable for me) of shell scripts or means of utilizing existing shell programs that I have written with the help of numerous online resources for my own projects. I value the shell for many reasons, one of which being that it assists dramatically in automation.

Automation can be an incredible boon to time and sanity savings, and reduction of human-induced errors that come from tedium. Automation however is not favorable in all situations, for example where innovation may be hindered. Many examples herein are used to automate tasks by performing a set-in-stone series of data conversions, calculations, executions, etc...; it is wise to consider when automation is beneficial and when other mechanisms for time savings such as improvements in processes may be more prudent. That being said, this text concerns mostly the automation of processes with the shell: an effectively automated user.

Sections in this text include general means of getting used to the shell, then how to execute commands and programs in the shell, and finally running your own shell scripts with all sorts of applications ranging from web scraping to making your own curses-style text-based user interface. All the code I supply is intended to be plug-and-play and should work out of the box. That being said however, I supply all of the following without any guarantee of any kind and without any warranty of any kind. Any ramifications of the use, deployment, or any other possible means of implementation or exploitation of the information contained within this text are the sole responsibility of the reader.

Nothing whatsoever within this text is meant to be direct or indirect or any form of legal advice in any capacity at all. Automated scripts can be used for good and innocuous things just as they can be used for malicious purposes. Don't be malicious, use what is in this text for good and for your own non-malicious purposes. Continuing to use this text constitutes an agreement on the part of the reader to use the contents of this text, the information it contains, and everything it references for no malicious or illegal activity. Furthermore this user agreement absolves the author in any and every capacity of what the end-user of this texts does or plans to do with the information that they gain while reading, studying, or implementing this text or the things it references.

In general terms, as it relates to computing, a shell is a user interface for an operating system's services. In order to view this document on a computer, you are probably using the explorer.exe shell (a fun bit of mischief in elementary school was to kill the explorer.exe processes on many networked computers...totally "hypothetically"...of course) if you are on Windows or the GNOME shell on Ubuntu Linux. These shells are graphical user interfaces (GUIs) and, chances are, you already know how to use at least one of them. This document will attempt to teach the reader how to use a terminal shell, like BASH in the stead of, or in addition to a GUI shell.

The primary focus of this document will be an introduction to command line use and scripting in the BASH shell for Linux and UNIX (specifically Darwin) machines. This guide is by no means meant to be exhaustive and absolutely no warranty is offered with this document.

As it stands, BASH is one of, if not the most widely used shell on Linux and UNIX machines. Using the shell from a terminal offers a user much greater access to the machine and its contents, especially when run as an administrator or a root user (the user on the computer which has full access to everything on the machine) than does someone using the default GUI shell.

In addition to the obvious benefits of controlling in relative totality the processes on a machine, uses for the shell include automating system tasks, accessing remote systems like supercomputers, running an incredible variety of programs, downloading programs from trusted sources and compiling them all through one terminal window.

1.1 Motivation for this text

Somewhere around my sophomore year in college, the mechanical hard drive of my very first (and about six year old at that point) laptop died spectacularly, taking my files and wildly generic super, super, super awful operating system which shall not be named, down to the clicky, crunchy, hardware failure grave. Intending to never again spend a penny on the operating system that shall not be named, I bought a new disk and hopped on the Linux train.

This text was written in small bursts over a period of several years with the intention, more or less, to be a simultaneous compendium of notes on BASH commands for my own reference and so I could get more comfortable writing in $\LaTeX$. I'm supplying this as a text for whoever my wish to use it for their own research work with absolutely zero warranty or guarantee of any kind. Any repercussions of the use of any and all code or written word given in this document are solely the responsibility of the party using or referencing this text and not at all the responsibility of the author. Use BASH and the shell for good!

I have done a reasonable amount to compile the following into cogent, brief, and 'recipes' for various useful operations in the BASH shell. I have a significant angle toward scientific computing and data analysis so that's what I present here. All reasonable efforts have been made to ensure that the supplied code runs smoothly, however the contents of this text are not guaranteed to be perfect and the user may have to modify some things slightly to fit updating systems or slightly different system architectures or programs (especially as things may change slightly with software updates and the like).

1.2 Acknowledgements

This text is dedicated to all those that lift me up and help to illuminate my path: my mother, father, brother, mentors, and friends. The following guides were written with the hope that they might be useful to others trying to use computational science techniques by reducing the somewhat formidable barrier to entry that I experienced when beginning.

I've learned mostly all that I know about BASH and code in general from a litany or resources too long to retell in this text and, unfortunately, too numerous to remember. However, some specific parties to thank are as follows: My friend Dr. Patrick E. Sims, a mentor to me since I was an undergraduate and he was working on his chemistry Ph.D. who is a wizard with the shell and first principles calculations. My parents who are lovingly responsible for a lifetime's worth of consistent support in learning and gaining skills. I also thank my brother Dr. Douglas G. Bopp for a friendly rivalry finding solutions to all manner of scientific, engineering, and computational problems; his hard work and dedication to solving complicated, exciting, and technologically relevant problems in physics and engineering, as well as how he's built his own business inspire me.

I acknowledge and appreciate the guidance of my thesis adviser Dr. Zhaowei Liu for his mentorship, and guidance on my research topics. I greatly appreciate the conversations, mentorship, and the access to super-computing resources that were given to me by my professor and thesis committee member Dr. Tod Pascal and the National Energy Research Scientific Computing Center which I utilized during my Ph.D. research and to grow my skills in BASH. I've also learned a lot of what I know about shell scripting from the developers and example-writers of the Quantum Espresso code, they influenced the way that I create and compose shell scripts for my own personal and professional use to a large extent. I use the Ubuntu operating system whenever I can (additionally, I use OpenSUSE and others but Ubuntu is my go-to). The Ubuntu creators sent me a free copy of Ubuntu 13.10 saucy salamander in the mail completely at their own expense on a live CD; that's a time for which I am nostalgic and grateful. The three years of support awarded to me by the Department of Defense (DoD) through the 2019-2022 National Defense Science and Engineering Graduate (NDSEG) Fellowship Program and the Air Force Office of Scientific Research (AFOSR) were of enormous benefit, affording me the latitude to complete this text and my PhD work. I would also like to thank the hard work and dedication of Allegra Swift from the UCSD Geisel Library who helped me pull together all of the resources for publishing this book.

It's very surreal to have completed the first version of this text since some incarnation of it has been living in my head for the better part of a decade. It's too strange for me to call this entirely complete, so let's call it a good start! The last years spent intermittently writing these pages have seen me through life in two states, the entirety of graduate school, many late nights, all sorts of uncomfortable major historical events, and such a proliferation of friendships, freedom, growth, and *becoming* that I just can't help but smile while writing this. Right now, it's one of those rare perfect moments and I'm delighted to be able to share it with you!

Some things and parties I've relied on and am grateful to thank are the following: La Jolla and its wild, misty, Seussical vibe of pristine beaches and confused-looking trees became my new home, a better place would be hard to find. Regent's Pizzeria with its deep dish slices for how it reminds me of an old home. 757, my delightful shoebox apartment at UCSD, for its cozy accommodations, surrounding community, and how it became my new home. My Crown Victoria, that beautiful old bulletproof black and white Arizona police interceptor, I am so pleased and happy it's kept me safe and given me the freedom of exploration over all these years.

Thank you everyone, my parents, brother, family, friends, mentors, and of course Tony Twenty Toes—the cat—you've bolstered me to not lose my courage and your contributions to who I am

and to how I see, as well as your friendship are valuable beyond what I can concisely express.

1.3 Acronyms defined

Acronyms are used in the text; to try and alleviate any confusion, I have tried to catalog most of those acronyms here:

^	Literally the key-press of the control key
^c	Interrupt or the 'kill' command
^r	The reverse-i-search
^M	The carriage return
&	The fork POSIX command
Å	Angstrom unit: 10^{-10}meter = 10nm = 1Å
a_0	The lattice parameter
arg	Argument
BASH	Bourne Again Shell
cat	Concatenate
coord	Coordinate
.csv	Comma separated values: a file extension
ctrl	Control: the literal keystroke of the control key
.dat	Data: a file extension
DFT	Density functional theory
diff	Difference
dir	Directory
DOS	Disk Operating System
EAM	Embedded Atom Model
EOF	End of File
fish	Friendly Interactive Shell
GUI	Graphical User Interface
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
lat	Lattice
MD	Molecular Dynamics
MEAM	Modified Embedded Atom Method
mpi	Message Passing Interface
opt	Option
param	Parameter
png	Portable network graphics: a file format
ps	PostScript: a file format
POSIX	Portable Operating System Interface
QE	Quantum ESPRESSO
recipr.	Reciprocal
seq	Sequence
su	Super User
sudo	BASH Command Meaning Super User Do (e.g. su do)
.svg	Salable vector graphics: a file format
tty	Teletype
TUI	Text-Based User Interface
TZ	Time Zone
UI	User Interface
uname	User Name
var	Variable
VASP	The Vienna Ab initio Simulation Package
w/	With
XRR	X-ray Reflectometry

1.4 BASH cheat sheet: my most common commands

For the impatient (like myself), here are a few of my most commonly used commands in the shell. I refer back to these all the time.

Compress a file with tar:

```
tar -czvf archive.tar.gz directory-or-file
```

Uncompress a file with tar:

```
tar -xvf archive.tar
```

Set common aliases:

```
alias whcih='which'; alias sl='ls'
```

Remove an alias:

```
unalias whcih='which'; unalias sl='ls'
```

Echo field 1 from row 1 in a text file into a variable with awk:

```
var=$(awk '{if(NR==1) print $1}' file.txt)
```

Check size of all files in a directory with du:

```
du -sh *
```

Connect to a remote system with ssh:

```
ssh -l uname -Y uname.uname.com
```

Copy a file from a remote system to a local machine with scp:

```
scp uname@uname.uname.com:~/file.tar.gz /Users/mainuser/Desktop
```

Copy a file from a local machine to a remote system with scp:

```
scp file.tar.gz uname@uname.uname.com:~
```

Switch commas for spaces using sed:

```
sed -i 's/,/ /g' ${Temp_File_Name}
```

Create a variable from math operations on n other variables with bc

```
Calc_Var=$(echo "scale=2;($a)*($b)" | bc)
```

Translate a file with dos line-endings into the UNIX format using tr:

```
tr -d '\r' < file.windows.csv > file.unix.csv
```

Execute a shell script and tee its output to a file:

```
./Script_Name.sh |& tee -a README.txt
```

Create a file with cat and redirects:

```
cat > File_Name.sh << EOF
#!/bin/bash
echo "Add some contents here"
EOF
```

1.5 The Bourne-again shell (BASH)

BASH is my favorite shell for many reasons including that it's what I learned to script with and am most comfortable using. There are many shells with many interesting and useful features. fish, and zsh (especially the ohmyzsh) packages are popular and friendly to work with. It is important to note that sometimes there are commands which will work perfectly in BASH but may not work well in sh or csh (which we will talk about in the next section). Making sure that your command line interpreter can actually execute the commands that you are giving it may sound trivial but it can be significantly frustrating if its an unknown unknown when you're just starting out with the terminal (this becomes vastly more important when scripting with the shell).

1.6 Accessing the shell

There are many shells, worthy of mention, beside BASH, are the following: sh, csh, zsh and fish (the former being one of this author's favorites especially for Darwin machines). Shells can be accessed through a command window called the terminal (sometimes called the console or similar on other operating systems). In Ubuntu Linux and Darwin UNIX, Terminal is the default command interpreter for the shell. Ubuntu users can easily access the terminal with the keyboard shortcut Ctrl+Shift+T or from the start menu; Darwin users can access the terminal from the spotlight search or the Utilities directory in their Applications directory.

Whatever way you access the terminal, the end result will be similar. To list what shell you are using in the terminal, type the following:

```
echo $0
```

As explained, this will return the shell you are using. In order to switch shells (assuming that they are installed) a user need only enter into the terminal the name of the shell they want to run. For example, if one wanted to run the c-shell, the user would need only type the following into the terminal:

```
csh
```

To exit this shell, back to the default shell, one only needs to enter the following:

```
exit
```

For the rest of this text, we will focus on the ubiquitous command line interpreter BASH because it is by far and away the most popular program of its type.

2 Shell Commands and How to Use Them

Running commands in the shell is made to be relatively straightforward, if you've been following this guide, then you've already successfully run several commands...no biggie eh?

Of particular and ubiquitous value is the manual command; as its name implies, 'man' will open the user manual for a specific command. For example, run the following:

```
man bash
```

As will be visible in the terminal window, you have been pulled into a program which shows the user available information for the BASH shell. Most programs installed on the the system that are accessible through the terminal should have a manual page. This author cannot stress enough how helpful this will be for a user in their future. Enter q to exit the manual page.

In the next sections we will discuss a list of several useful tools for successfully interacting with the BASH shell (and most other shells for that matter). Do not be daunted by the several pages that these sections will occupy, all of the tutorials contained in this section are relatively straightforward and shouldn't take much time to complete. Much of what is contained in this section may be redundant to a more skilled shell user and such a person my be better served by skipping on to subsequent sections. However, the examples that follow in this chapter are included for reference and completeness.

After completing this section, a user should be able to do all of the following: navigate their system in the shell; create, open and delete files and directories (as well as archives); read, write and destroy files; set and change the ownership or the mode of a file; use and identify the usefulness of special characters; and run programs as the root user or the fake root user (a command whose level of usefulness is devious).

At any time, it is helpful for the user to know the following: first, the command:

```
!!
```

will repeat the last input command to the terminal; second, using the up and down arrows in the command processor mode of the terminal will navigate a user through the history of recently issued commands. Pressing the up arrow once and then enter will have the same effect as running the '!!' command. Additionally, using the tab key will tell the shell to attempt an auto completion of what you are typing; for example, if your directory contains a file called user.txt and one were to type 'us' and then press tab, the shell would attempt to insert the remainder of the file name, assuming that there are no other files starting with 'us.'

2.1 Shell navigation: pwd, cd, ls, file

In order to navigate your way around in the shell, you need to use text commands, just like almost everything else in the shell. There are, however workarounds for the navigation explained herein, these will be explained at the end of this section.

Enter into the terminal the following:

```
pwd
```

This program is called 'print working directory' and it will do just that. BASH will output the location of the directory in which you are currently located. This will be a /home/user directory for an Ubuntu user or a /Users/macuser type directory for a Darwin machine.

In order to navigate one directory closer to the root directory, enter into the terminal the following:

```
cd .. ; pwd
```

You have just run two commands one after the other: cd .. navigated you 'up' one directory; pwd lists for you your new location. Now, enter the following into the terminal:

```
cd /
```

You are now in the root directory, this can be evidenced with pwd at any time. Using the cd / command is similar to the bare cd command which sends you to your home directory. Let's say that you're a curious user and you want to see what files and folders are in your root directory. Enter the following into the terminal:

```
ls -a
```

What have you done? All together, you have listed all of the directories and files in the root directory. Now try running the command above again but without the '-a' tag. You'll notice that a variety of items which were prepended with a '.' are now missing. These are called hidden files and hidden directories, the '-a' tag has allowed you to list all of these items, regardless of their having been prepended with a '.' or not.

How do you know what a certain file or folder is, especially if your terminal does not highlight types with different colors? Fear not, the following command will never fail to help:

```
file bin/
```

You have just executed the 'file' command on the bin/ directory. The BASH output should be something like: "bin/: directory." In order to navigate into one of the directories which you can see, it is as simple as this:

```
cd /bin
```

List out the files and you can see that you are in a directory which contains programs that can be run with the shell. Do you see our friends ls, pwd and BASH? With the cd command, navigate back to the home directory and enter again into the command window the following:

```
cd
cd /bin
```

With pwd you can see that you are again in the /bin directory, in the same way, you can access any directory in your computer (assuming that you have the correct permissions) with the cd command.

Your home directory is perhaps the most frequented by the user. One final usage of cd or navigation in general is with the tilde ~. If, say, you need to specify the location of a file or a directory for the shell, you can use the entire path to the file or directory like so (here, all we are doing is creating a variable whose value is the location of an executable, vasp in this case. We'll talk more about variables a little later on in this text):

```
Program_Location="~/uname/codes/vasp/vasp.5.3/vasp"
```

However that becomes cumbersome and inconvenient for instances where your home directory is several levels inside of other directories. If, for example, your home directory is named 'uname', then you could change the above location to the below location, the two are equivalent but one just assumes that /uname is the home directory of the user

```
Program_Location="~/codes/vasp/vasp.5.3/vasp"
```

The only difference between the previous two commands is that one has had the string /uname truncated into ~which is very handy for all sorts of things when you don't want to type out the full path every time.

There are other ways to navigate and explore files in the terminal than with the cd command, one such shining example is a program called Ranger. Ranger is a file explorer written in a language called ncurses which allows for extremely efficient exploration of files and directories on your system. To run Ranger, you need to install the program, if you are an Ubuntu (or similar OS) user, this can be accomplished easily with the following command:

```
sudo apt-get install ranger
```

(sudo is a command called supervisory user do, it is extremely powerful and can cause unintended consequences on your computer's system if used incorrectly. Don't let the power of sudo deter you though! (We'll talk more about it later). If you are a Darwin user, then you will need to enter the following:

```
brew install ranger
```

Brew is part of the homebrew package manager for Darwin which is similar to apt-get; to install it, search the internet for the Homebrew Package Manager and follow its instructions. Other systems, such as distributions based on Fedora Linux and Red Hat will need you to substitute the 'apt' command for the 'yum' command.

Ranger is a lot of fun to use and can be really helpful when attempting to navigate through many levels of directories or look for a specific file in many directories or among many other files (when that is you're not using a search command like find which is discussed in section 3.5). Ranger gives you a more gui-like experience compared to the sort of cold-feeling use of ls and cd. You can run ranger (after installing it of course) with the following command:

```
ranger
```

I highly recommend taking some time to use ranger to explore the filesystem and see first-hand some of its capabilities! You can even use it to preview files and their headers which I find to be enormously useful, especially when using remote machines.

2.2 Handling files in the shell

Fast, flexible, modular, automatable management of the files that contain our data and calculations is one of the most attractive features of the shell. We will see in this book how the shell is similar to an automated user that never makes mistakes and will perform repetitive tasks like file manipulation, handling, copying, comparing, searching, etc... expertly and astronomically quicker than could a human user; that's the true power of shell scripting. More or less, everything contained within this book could be done without the automation that BASH and other shells allow, however the value of this text is in the time savings that come from the sheer intractability of trying to do some of those operations 'by hand'.

In the following sections, we will take the first steps into some of this by exploring ways that files can be handled and manipulated in the shell.

2.2.1 Create, rename, copy, move, and delete: touch, mv, cp, rm

Using the shell, it is facile to create a file (let's call the file file.txt); this can be done with the following command:

```
touch file.txt
```

Upon checking the contents of the current directory (remember the ls command?), one will find that the file called file.txt now exists. Now try running the following command:

```
mv file.txt .file.txt
```

Listing again the files in the directory will not show the file, adding the '-a' tag to the 'ls' command will reveal the file—you have modified the file by renaming it; additionally, you have turned it into a hidden file. The command can be reversed to rename the file to what it was previously. This is one of the two uses for the move (mv) command in the shell. Renaming the file back to file.txt will be left as an exercise for the reader.

Issue the following command to the terminal:

```
mkdir text_files
```

List the contents of your current directory and you will see a new directory called text_files (the _character is used because spaces in a file name are tedious to call in the shell, one would need to subpend each separate word in the file name with another backslash if one wanted to included spaces in a file name).

Now issue the following command:

```
cp file.txt copy.txt
```

Listing the files in the directory, a user will find that there is a new file called copy.txt; this is the power of the cp command, to make copies of files and name them in the same line of text.

Enter the following command:

```
mv file.txt text_files
```

Again list the items in your current directory and you will see that the file.txt file is no longer present. You have in fact moved the file into the previously created directory. This is the second use of the move (mv) command. In the command, the target directory can be any which you want (assuming that you have the proper permissions).

Navigate into your newly created directory and issue the following command:

```
rm file.txt
```

Upon listing the items in the directory, one will find that there are no contents. What you have done is use the remove 'rm' command to remove the file. It is worthy of note that the 'rm' command does not overwrite or destroy the file, it only erases the link between the file and the shell so that it is difficult (but not impossible) to access. File destruction techniques will be explained in section 2.2.2.

Navigate back to the parent directory (try using 'cd ..') and list its contents, you should again see the text_files directory. Now issue the following command:

```
rm -r text_files
```

List again the items in your current directory and you will see that the text_files directory is missing. What you have done is erase the text_files directory. You used the '-r' tag because the vanilla 'rm' command will not erase files and directories recursively.

2.2.2 Reading, writing, and destroying files: ls, nano, cat, head, tail

When a user spends time in the shell, they may find it convenient to view and edit text in files, as well as securely deleting them (shredding or destroying). There is a program called 'sed' which allows a user to edit text without another program to aid them, 'sed' is beyond the scope of this section but is extremely useful and will be discussed at length in sections like 5.3 and 5.7.

There are two primary quick ways to view text, only the first of which will allow you to edit. The first of these is the use of text editors (like text edit on Darwin systems or gedit on Ubuntu and similar systems), the second is the use of operations which read text from a file and print that text to the terminal.

Find for yourself, if you will, a text-containing file whose length is considerable, maybe more than 40 lines and move or copy it into your current directory (let's call the file text.txt).

Let's try to edit the file called text.txt; to do this, enter into the terminal the following command:

```
nano text.txt
```

In your terminal, you should see the entire body of text contained in the file called text.txt, additionally you will see a list of possible commands at the bottom of the screen and the label 'GNU nano' at the top left.

Currently, you are in the terminal-based text editor called nano. You can read, write to and delete from the document. Enter ctrl+o to save changes and ctrl+x to exit the nano program (in the shell, the control key is abbreviated as '^' so, ctrl+o would be abbreviated as '^o').

There are many other programs which are made to edit text, such as pico (closed source), vi and vim and emacs. These programs are beyond the scope of this section but vim specifically will be discussed at greater length in section 3.7 because of how ubiquitous and useful it is.

Alternative to nano, there are many ways of viewing text without entering another program or editing the text—helpful when one wants to just view the contents of a file. Enter into the terminal the following commands:

```
cat text.txt ; head text.txt ; tail text.txt
```

In your terminal, you should have seen first the entire text of the file printed out, then the top ten lines of the file and then the bottom ten lines of the file; these are, respectively, the functions of the cat, head and tail commands, all of which are extremely helpful.

Finally, it may be necessary for the purpose of security, to be able to securely delete files: not just to remove the link which allows the shell to access the and mark the to be overwritten by new files. This can be useful when data perhaps for measurements that you are very near to publish or patent and that are stored on publicly accessible hard disks. There are several programs which can do this effectively, but the Linux shred command seems like a reasonable choice.

Let's again use the example of a file called text.txt. Make sure that such a file is indeed in your current directory and run the following command:

```
cp text.txt delete.txt ; rm -Pv file.txt
```

Above, you copied the file text.txt, naming the copy delete.txt. Next, you ran the rm command with the 'P' tag (overwrite files before unlinking) and the 'v' tag (verbose). It is unknown to this author how effective the 'P' option is relative to the two commands which will follow (although it is assumed to be less effective).

On Linux machines, chances are good that the program called 'shred' will be either installed or easy to install from the terminal. On Darwin machines, the program called 'srm' should come pre-installed with the machine.

Make again the text.txt (instead this time naming it delete.txt) file with the same commands as above.

Now, if you are running a Linux operating system, run the first of the two following commands; if you are running a Darwin machine, run the second of these two commands:

```
shred -uvz delete.txt
```

```
srm -mvz delete.txt
```

In both cases, the file called delete.txt will have been overwritten, written completely with zeros and then unlinked. In both cases, the 'v' option was added so that the shell would print out what it was doing (this is known as the verbose mode) and the 'z' option was added to write the file completely with zeros before truncation.

It should be known that, with sufficient time and computing power, these files can be reconstructed, the time and power necessary for such a reconstruction being directly proportional to the number of overwrite steps and time spent overwriting. It is beneficial for the reader to explore the manual pages of these programs before their use.

2.2.3 Viewing filters and file comparisons: more, less, and diff

Viewing files in the shell can be done in numerous ways, some of which (including tools for editing text) were just discussed in section 2.2.2. However, there are many more ways of viewing files which do not involve editing the text like might be done with nano, vim (discussed in section 3.7), vi, or listing out the entire contents of the text to the terminal like would be done with the 'cat' command. Two such programs are what are known as viewing filters and are named 'more' and 'less'.

These viewing filters allow for viewing a file as if it were split into pages on the terminal, this variety of viewing is known as paging. The 'more' command lets you page through a file one full screen at a time (e.g., the file is printed to the terminal as discrete chunks that you can only navigate discretely between as if you were turning the pages of a book). The 'less' program is frequently explained as being the opposite of 'more' and allows for paging with scrolling where it is easier to navigate smoothly up and down a file.

We can demonstrate the more program using an INFO.OUT file from an Exciting (see chapter 7 for more about the Exciting DFT code) with the following command:

```
more INFO.OUT
```

The result will look something like the following where 'more' shows you the contents of the file and also shows the percent progress through the file that you are currently viewing. However, more is significantly irritating in that it will only allow you to advance one line at a time by using the enter key (or with special commands that allow you to skip forward some number of lines, etc...), and it will print the text file to the standard output of the terminal. The manual page for more explains that the program is "...especially primitive" which is absolutely hilarious to this author. Please view a sample output of the more command in the following terminal session (you can quit more any time by striking the q key):

```
================================================================================
| EXCITING OXYGEN started                                                      =
|                                                                              =
| compiler: ifort (IFORT) 19.0.3.199 20190206                                  =
|                                                                              =
| MPI version using      32 processor(s)                                          =
| |  using MPI-2 features                                                         =
|                                                                                 =
| Date (DD-MM-YYYY) : 11-08-2021                                                  =
| Time (hh:mm:ss)   : 06:07:09                                                    =
|                                                                                 =
| All units are atomic (Hartree, Bohr, etc.)                                      =
===================================================================================

*********************************************************************************
* Ground-state run starting from atomic densities                               *
*********************************************************************************

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Starting initialization                                                       +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 Lattice vectors (cartesian) :
      0.0000000000      4.0015000000      4.0015000000
      4.0015000000      0.0000000000      4.0015000000
      4.0015000000      4.0015000000      0.0000000000

 Reciprocal lattice vectors (cartesian) :
     -0.7851037495      0.7851037495      0.7851037495
      0.7851037495     -0.7851037495      0.7851037495
      0.7851037495      0.7851037495     -0.7851037495

 Unit cell volume                           :      128.1440540068
 Brillouin zone volume                      :        1.9357137978
--More--(4%)
```

Unlike the more program, less is actually fantastic. This is factual because 'less' has vastly enhanced functionality on top of the basic idea of the 'more' program. Some of this functionality is that you can scroll with your mouse through the file with numerous more options and capabilities. Additionally, 'less' will not print the contents of the file to the standard output in the terminal: this feature is especially important when you don't like to have a cluttered terminal, don't want to constantly be scrolling between different portions of the terminal session, or just hate having to clear the terminal constantly to eliminate the extraneous standard output. We can demonstrate the less command with on the same INFO.OUT file by using the following command (you can exit from less at any time by striking the q key, as is customary in many BASH programs):

```
less INFO.OUT
```

The terminal will switch to a new view where less will populate the entire window with the contents of the file that you have directed it to as well as giving an indication of the file name and the percent progress through the file that you have made in your viewing printed to the bottom of the terminal.

```
===================================================================================
| EXCITING OXYGEN started                                                         =
|                                                                                 =
| compiler: ifort (IFORT) 19.0.3.199 20190206                                     =
|                                                                                 =
| MPI version using      32 processor(s)                                          =
| |  using MPI-2 features                                                         =
|                                                                                 =
| Date (DD-MM-YYYY) : 11-08-2021                                                  =
| Time (hh:mm:ss)    : 06:07:09                                                  =
|                                                                                =
| All units are atomic (Hartree, Bohr, etc.)                                    =
==================================================================================

**********************************************************************************
* Ground-state run starting from atomic densities                               *
**********************************************************************************

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Starting initialization                                                        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 Lattice vectors (cartesian) :
      0.0000000000      4.0015000000      4.0015000000
      4.0015000000      0.0000000000      4.0015000000
      4.0015000000      4.0015000000      0.0000000000

 Reciprocal lattice vectors (cartesian) :
     -0.7851037495      0.7851037495      0.7851037495
      0.7851037495     -0.7851037495      0.7851037495
      0.7851037495      0.7851037495     -0.7851037495

 Unit cell volume                           :      128.1440540068
 Brillouin zone volume                      :        1.9357137978
INFO.OUT lines 1-34/783 5%
```

Another useful command in BASH is diff and, as the name might imply, it allows you to see the differences between files more easily than painstakingly searching through the files by yourself with your human eyeballs. The diff command is demonstrated here and used to view the differences between two scripts that have very similar purpose (which will be discussed in chapter 9. Here we compare two files VASP_EV_Collector.sh VASP_Slab_EV_Collector_v2.sh with the diff command. The standard output then consists of pairs of lines that are given where some difference is detected between the two files on a certain line in both of the files. Please see the following terminal session:

```
uname@uname:~> diff VASP_EV_Collector.sh VASP_Slab_EV_Collector_v2.sh
4c4
< #-:-:-:-:-:-:-:-:-:-:-:-:-:-:- VASP_EV_Collector.sh :-:-:-:-:-:-:-:-:-:-:-:-:-:-
---
> #-:-:-:-:-:-:-:-:-:-:-:-:- VASP_Slab_EV_Collector_v2.sh :-:-:-:-:-:-:-:-:-:-:-:-
7c7
< #-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:- October 6, 2021 -:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
---
> #-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:- October 5, 2021 -:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
14c14
< # File_Name="VASP_EV_Collector_v3"; touch ${File_Name}.sh; chmod +x ${File_Name}.
    sh; vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README_EV_Collector.txt
---
> # File_Name="VASP_EV_Collector_v2"; touch ${File_Name}.sh; chmod +x ${File_Name}.
    sh; vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README_EV_Collector.txt
33c33
<    echo "  Entering directory $a"
---
>    echo "Entering directory $a"
36,38d35
<       Convergence=$(awk '/F=/{print $1}' OSZICAR)
<      if [[ $Convergence = 1 ]] ; then echo '  Converged' ; else tput bel && echo
    '<!> Warning: Calculation Did Not Converge <!>' ; fi
<
49c46
<      echo "  For lattice parameter = ${a0}, Total energy = ${E0} Energy per ion = $
    {Energy_Per_Ion} Volume per ion = ${Volume_Per_Ion} "
---
>      echo " For lattice parameter = ${a0}, Total energy = ${E0} Energy per ion = ${
    Energy_Per_Ion} Volume per ion = ${Volume_Per_Ion} "
uname@uname:~>
```

The diff command is especially useful in version control of documents and to highlight where changes may have been made without other cumbersome implementations of the same functionality.

2.2.4 Create and unzip archives: tar, and gzip

Here we will see how to store and extract files from a .tar archive as well as use the programs called gzip and gunzip. The discussion here will be limited to the .tar (tape archive) files and the .gz and .bz2 compression formats. As always, the reader is encouraged to explore the manual page for tar; this can be easily reached with the following command:

```
man tar
```

The (tape archive) .tar.gz file format is ubiquitous for packing files, especially on Linux systems. documentation for the tar program is excellent and straightforward on the internet. As a quick primer, one can create a .tar.gz archive from a single or a number of files with the following command

```
tar -czvf name-of-archive.tar.gz /path/to/file(s)_to_archive
```

Here, the -c option means to 'create.' To unpack files from the same archive, one can issue the following command (assuming that you are located in the same directory as the archive of interest):

```
tar -xvf name-of-archive.tar.gz
```

Additionally, sometimes it just becomes annoying to have all the contents of the archive printed to the terminal as it is being unpacked because you may have pertinent information already printed to the terminal that you don't always want to scroll up to see. To suppress the output of tar, you can remove the -v (verbose) option and run the unpacking as follows:

```
tar -xf name-of-archive.tar.gz
```

To pack files into a .tar package, it is as simple as issuing the following command (omit the above if you wish, substituting files for what ever is necessary):

```
touch file1.sh file2.sh file3.sh
tar -cf archive.tar file1.sh file2.sh file3.sh
```

If you list out the contents of your directory, you will see that there are four new files, one of which being appended with the .tar format; this file is the package which you have just made. To zip the file, one must only issue the following command:

```
gzip archive.tar
```

Listing the contents again will show that the file has again been appended with a new format, this time its name will be archive.tar.gz; the .gz format is that of the gzip program.

To undo all of the commands which you have just done, issue the following commands:

```
rm file1.sh file2.sh file3.sh ; gunzip archive.tar.gz ; tar -xvf archive.tar
```

What you have done is the following: first, you deleted the three original files using rm; next, you unzipped the .gz file with the program called gunzip; finally, you unpacked the .tar archive (adding the -v tag will print the operations to the terminal) and got back all of the three files which you made in the beginning. Listing the contents of the directory will confirm this.

If (as you will in the future of this document) you are to run into a file appended with the format .tar.bz2, you can use the same program (tar) to unpack the files, you must just add the -j tag to the list of options you choose; an example is included here for a theoretical file called archive.tar.bz2:

```
tar -xvjf archive.tar.bz2
```

2.2.5 Z-commands: zcat, zdiff, zless, and zmore

If you have ever used the archive tool in Ubuntu, you may have noticed that is has a very appealing functionality in that you can use it to view the contents of archives without actually unpacking the archive file. This is an especially attractive feature when you use archives as a large part of your workflow and/or have a proliferation of very large archives that you use to store your important files and calculations. BASH has several programs with similar effects which are generally known as the z-commands. These commands operate on archives generally ending in *z (e.g., *.tar.gz, etc...) but can be given options which allow then to operate on other files beyond those in .gz and similar formats.

We can see the use of zcat in the following commands where we create a text file and then pack it into a *.tar.gz archive and try to concatenate its contents with the cat command. We see however that trying to use the cat command returns a standard output that is not human-readable. This can be resolved by using zcat instead. This is especially useful for VERY large text files like what you might find in a .out file for a large DFT or MD computation. Please see the following terminal session:

```
uname@uname:~> touch hello.txt
uname@uname:~> echo hello > hello.txt
uname@uname:~> gzip hello.txt
uname@uname:~> ls
hello.txt.gz
uname@uname:~> cat hello.txt.gz
??nahello.txt?H???? 0:6
uname@uname:~> zcat hello.txt.gz
hello
uname@uname:~>
```

Using the zcat command here allowed us to read the archive contents without actually unpacking it! This works great for individual files but can get weird for archives of many files and directories.

Using the pipe operation (please see section 4.2 for more coverage of the pipe), we can reroute the standard output of zcat into the BASH command more:

```
zcat archive.tar.gz | more
```

This is equivalent to the simplified command:

```
zmore archive.tar.gz
```

Similarly, we can pipe zcat into the BASH command less:

```
zcat archive.tar.gz | less
```

Which, similarly, is equivalent to the simplified command:

```
zmore archive.tar.gz
```

We can also use the command zdiff to see the differences between archives. Please see the following terminal session (which builds upon the previous example with the hello.tar.gz file):

```
uname@uname:~> touch goodbye.txt
uname@uname:~> echo goodbye > goodbye.txt
uname@uname:~> gzip goodbye.txt
uname@uname:~> diff hello.txt.gz goodbye.txt.gz
Binary files hello.txt.gz and goodbye.txt.gz differ
uname@uname:~> zdiff hello.txt.gz goodbye.txt.gz
1c1
< hello
---
> goodbye
```

The standard output here shows that there is a difference on the first line in the first field where the first file says 'hello' and the second file says 'goodbye'. Obviously this is a trivial case and is vastly more helpful for large files that would make your eyes bleed to directly compare, especially after a long day or night working on the terminal. Tools like these make life easier.

2.2.6 Estimate a file's disk usage: du

Using shell programs that write files is common practice, being able to quickly see the size of a file that has been created by the shell or a pre-existing file is useful. The shell program du can help with this. Run the command:

```
du -sh *
```

The output of du may look something like this (ls is run here first to list quickly what is in the directory of interest):

```
uname@uname:~/codes> ls
Pseudo  atomsk_0.11  evfit  exciting_mpi  exciting_serial  exciting_smp  sources
    vasp
uname@uname:~/codes> du -sh *
126M  Pseudo
21M   atomsk_0.11
93K   evfit
278M  exciting_mpi
256M  exciting_serial
257M  exciting_smp
70M   sources
178M  vasp
```

du has listed all the files contained in the directory as well as their sizes (e.g., K for kilobytes, M for megabytes, G for gigabytes, ...)

2.3 Special characters: :, ;, &, &&, |, $, !, <, >, *, ~, ?, /, \, %, ., .., ...

BASH utilizes quite a few special characters most or all of which are useful for properly understanding what is going on between different commands and in different parts of the shell environment. In the following sections, we will discuss some of these characters, their implications, and how (in some cases) they can be used to our great advantage.

Special characters in bash may encompass more than just a single character as well. In some cases a single character may be a special character, in some other cases repeating that character twice may be a distinct special character with its own uses and value distinct from the single character. We have already seen an example of this in the !! command which can be used to repeat the immediately previous command.

Since special characters are used in most places and very frequently in even the simplest of commands, it is difficult to create an individual section of special characters before other sections covering different functionalities of BASH that may be more intuitive to being with. However, there are some special characters that have very interesting and unique functionalities that fit well into this category and an attempt will be made here to create a section devoted to describing the special characters or linking to other sections that have good coverage of the special characters. Even though some of these special characters may be described in less depth elsewhere in this text, a table of BASH special characters and their meanings follows, I include this as a handy reference to the time of paging back and forth around this text trying to remember what's what.

Character	Character Name	Character Description
:	The null operator	See section 4.3.3 for an example of using the colon character with a while loop.
;	Command separator	Separates commands without condition and allows for stringing of multiple commands into one single command
&&	Conditional Command Separator	Separates commands and allows for the next command to be executed only if the previous command was executed properly
&	Fork	Executes all preceding commands in the background
|	Pipe	Allows the user to take the output of one program and turn it into the input of another program. See section 4.2 for more on pipes
$	Variable	Special character in the shell denoting a variable e.g., you could create a variable with a=1 and then call that variable with echo $a
!	Logical Not	The logical not symbol usually for use in conjunction with tests and square brackets
<	Redirect Input	Allows you to direct a file into a preceding command. See section 4.1 for more information on redirecting an input
>	Redirect Output	Allows you to take the output of one command and directly write it into a file. See section 4.1 for more information on redirecting an output
*	Sequence Wildcard	The sequence wildcard built-in. See section 2.3.4 for more examples
?	Single Character Wildcard	The single character wildcard built-in. See section 2.3.5 for more examples
/	Slash	Appends the names of paths in BASH. See section 2.3.1 for more information
\	Backslash	Also allows you to escape a built-in. See section 2.3.2 for more on the backslash
~	Home Directory	The tilde denotes the location of the home directory in Linux and UNIX systems
%	Modulo	Returns the remainder of an integer division operation. See section 4.3.1 for an example that includes modulo
.	Current directory	Special character denoting the working directory. Use the command pwd at any time to see the full path to your working directory
..	Parent directory	Special character denoting he parent directory e.g., the directory one level above the working directory
((...))	Double Parenthesis	Allows arithmetic expansion and evaluation. See section 4.3.1 for an example including the double parenthesis

2.3.1 The slash character /

The slash character / is interesting and has a few operations within BASH. First of all, it is used to denote a path e.g., a directory called directory is technically named directory/

Running the following command will print all of the files in the directory to the terminal and all of the directories will be printed with a slash following their name

```
ls -CHGp --color *
```

2.3.2 The backslash character \

Another use of the slash character / is to escape a built in operation. This is very useful if you want to actually print $a to the terminal in the event that you have a variable named a and not just print its stored value. Please see the following terminal session for a visual explanation:

```
a=1
uname@uname:~> echo $a
1
uname@uname:~> echo \$a
$a
uname@uname:~> echo "\$a is $a"
$a is 1
```

2.3.3 The modulo character %

The modulo operation, given by special character %, returns the remainder of an integer division operation. This can quickly be paired with the non-floating point arithmetic builtin expr as in the following example which uses expression to divide 11 by 6 and return the remainder:

```
uname@uname:~> expr 11 % 6
5
```

Here, the command has returned 5 as the remainder of 11/6 which itself is 1 with remainder 5. Similarly, we can see the benefit of the modular arithmetic (which can be used as a sort of periodic counter) in the following where we evaluate the remainder of 13/6

```
uname@uname:~> expr 13 % 6
1
```

The value that BASH has returned is 1 which makes sense because 13/6 is 2 with remainder 1.

2.3.4 The sequence wildcard character *

The wildcard character * is a way for the user to define the argument of a shell command more generally. * is extremely, massively useful to the point that I use it nearly constantly (mostly because it is very tedious to constantly type and retype commands and specific file names in the shell; * helps mitigate that issue. That is to say the wildcard adds an element of 'whatever' into a command. If it pleases the reader, run the ls command and then run the following:

```
ls *
```

The reader will find that the common ls command did its job and showed the (un-hidden) contents of the home directory. Of more interest is what happened when the wildcard character was added to the command: contents of the current directory are listed as well as the first order contents of all the directories in the working directory.

Next let's try making several files with different formats. Consider using the following (it's going to be helpful to run this command in a new, empty directory, for simplicity let's call it 'directory'):

```
touch {a..c}.txt {a..c}.dat {a..c}.sh ; ls
```

You've created nine new files of three different extensions using the touch command and then listed the contents of the working directory. But what if there were an arbitrary number of items in the directory and only the ones starting with a certain letter or ending with a certain extension are relevant? At the reader's leisure, run the following:

```
ls a*
```

```
ls *.sh
```

```
ls *.*
```

The terminal will print first a list of all the files starting with the letter a (lower case exclusively) regardless of any information following the first character. Next, the terminal will print a list of all the files that are .sh extended regardless of any information before the .sh. Finally, the terminal will list all of the files in the current directory. A discerning reader can tell that the wildcard operation is exceptionally handy when performing operations on multiple objects in the same command. Moving, removing, editing, and much more are accessible with the wildcard. All of the contents of a directory can be deleted with the sequence wildcard character (be careful to perform this operation only in a directory that has no files that the reader would miss if they were deleted), try the following:

```
ls ; rm * ; ls
```

The reader will find, after looking at the obvious differences between the lists given before and after the 'rm *' operation that there are no longer any files in the working directory. If the reader had executed mv * .. instead, then all of the files in the working directory would be moved one directory up in the tree (assuming that such a directory exists).

2.3.5 The single character wildcard ?

Similar to the sequence wildcard operation described in section 2.3.4 , the single character wildcard ? will look only for a single matching character. Please see the following terminal session (be not daunted, for the for loops used here are discussed in greater detail in section 4.3.1):

```
uname@uname:~> for a in {0..1000..500}; do mkdir $a_Dir; done
uname@uname:~> for a in {5..1005..500}; do mkdir $a_Dir; done
uname@uname:~> ls
1000_Dir    1005_Dir    1_Dir    500_Dir    505_Dir    5_Dir
uname@uname:~> ls ?_Dir
1_Dir:

5_Dir:
uname@uname:~>ls ???_Dir
500_Dir:

505_Dir:
uname@uname:~> ls ????_Dir
1000_Dir:

1005_Dir:
```

As can be seen, we have greater control over the specificity with which guesses are made by the terminal when using the ? character compared to the * character. In all honesty however, I do find that the * is more useful in the general cases but ? is included for completeness and reference.

2.4 Ownership and file modes: chmod and chown

If, say, someone were to want to set ownership parameters for a certain file, let's call it file.txt, then that person would need to use the change ownership function which is a built-in for the BASH shell. Enter the following into the terminal:

```
touch file.txt ; sudo chown uname file.txt
```

What we have done is the following: first, we created a file called file.txt; next, we modified the ownership parameter (with the program called chown) of the file such that the user called 'uname' is set as the owner of the file.

Ownership is important on a system for several reasons, one of the most important is that many system files are set to be owned by the root user (discussed in section 2.5), by default a current user will not find themself logged into the root account unless several steps have been taken. Modifying root-owned files and directories is generally impossible unless one is logged into the root account or one uses the sudo (to be discussed in section 2.5) command. Modifying the ownership of such a file can make modifying it simpler.

Change mode (chmod) is another program which the reader will find to be of enormous value in the coming sections (mainly in executing their own or downloaded scripts in the shell). Change mode is the program called chmod and it is used to change the file mode or the access control list. That is to say, chmod allows a user to define what users can do what with a file. Run the following command in the terminal:

```
touch scrip.sh ; chmod 755 script.sh ; ./script.sh
```

Here is what you have just done: the first section (as you should know from what was discussed previously in 2.2.1) created a file called script.sh; the next section changed the mode of the file to 755, this means that the owner can execute the file, read the file and write to the file but other users can only read and write to the file; finally, you executed the file called script.sh (using the ./ command will execute a file).

Congratulations, you have just made your very first shell script; while it may not have done anything particularly world-changing, it is the first step to shell scripting—an extremely powerful tool with which the bulk of this text is concerned. The same could have be accomplished if the reader had replaced the 755 with +x where x is the executable tag. To demonstrate the similarity between 755 and +x, the reader can compare the result of the previous command to that of the following.

```
touch scrip2.sh ; chmod +x script2.sh ; ./script2.sh
```

2.5 Superuser operations: su, sudo and fakeroot

Often, when running programs or commands in the shell, one may find that only users with administrator (or root) permissions on the system will be able to perform certain commands or access certain files or directories. For the purposes of security, it is ill-advised for a user to be signed permanently into the root account, so the question becomes: how do I run administrator commands?

One will find that the command to operate as the root user is called 'sudo,' an acronym for 'supervisory user do' (some people pronounce it like 'sue-do,' which is the technically correct way but fun people say it like 'pseudo'). Chances are that, if you are running a Linux system, you have already used the sudo command for installations. Try entering into the terminal the following:

```
sudo -i
```

The terminal will require the administrator password; when it is entered, the user will find that they have been logged into a new account called root@user where user can be replaced by the hostname of the user. This is, however, not usually recommended because of security issues and possibilities of damage to the system by a less experienced user or even simple mistakes. To log out of the root account, enter the following into the terminal:

```
exit
```

Running commands with root privileges without being signed into the root account is simple and can be accesses with the sudo command in the following way:

```
sudo echo "You're running this as the root user!"
```

For those intending to run interactively as root, that is to be free of the relative burden of entering the sudo command every time that something needed to be run as the root user, one can use the 'su' command. It's really just as easy as keying in the following command and giving your password, however the su command is generally considered much riskier to run that the sudo command. There are a lot of benefits of using sudo over su but for the general user, its mainly a concern of the extra time it takes to type sudo every time you run a command which can make changes which may be difficult or impractical to reverse giving the user more chance to contemplate whether they're doing exactly what they want to do. The exit command will again remove the user from the root mode.

```
su
```

A similar command to sudo is called 'fakeroot.' A user will find that the fakeroot command does not come pre-installed on most system;, use the appropriate command to install it now. Run the following:

```
fakeroot
```

A clever user will find that they have been logged into an account which is analogous to the root account but has no actual root permissions. The devious beauty of fakeroot is that, for many operations, it is enough for the system to just look like the user is the root user instead of actually being signed into the root account.

Exploration of the manual page for the fakeroot command will show its myriad uses which will not be enumerated here, suffice it to say however that there's a lot of fun to be had with fakeroot (however, only use it for good!).

2.6 Shortcuts in the shell

I'm a lover of most things that end up saving me time. As it turns out, many shrewd people feel the same way, to the extent that there are quite a few bits of time-saving functionality built into the shell. Some of these functionalities are listed in the following sections:

2.6.1 Quick-searching the BASH history: reverse-i-Search ^r

A particularly useful command in the shell is run with ctrl+r. This is what is known as the reverse-i-search and it allows you to quickly search through previous commands that you have submitted through the terminal to BASH. Entering ctrl+r into a BASH terminal session will result in your name in the terminal session switching to a line displaying the string reverse-i-search and some other characters like in what's shown below

```
MainUsers-iMac:~ mainuser$
(reverse-i-search)`':
```

If you begin to type into the reverse-i-search, you will see guesses that BASH presents to you based on how well the string you submit fits a string stored in the log of commands that you have submitted to BASH. At any time, the reverse-i-search can be aborted with the ctrl+c command. What follows are several search phrases that the reverse-i-search considered, the search terms are shown between the back tick and the apostrophe following the (reverse-i-search) phrase, and the reverse-i-search best guess is given after the colon:

```
MainUsers-iMac:~ mainuser$
(reverse-i-search)`s': ls
MainUsers-iMac:~ mainuser$
(reverse-i-search)`ss': ssh -l uname -Y uname.uname.com
MainUsers-iMac:~ mainuser$
(reverse-i-search)`sc': scp uname@uname.uname.com:~/Xpspeak41.zip /Users/mainuser/
    Desktop
MainUsers-iMac:~ mainuser$
```

Reverse-i-search is a convenient tool for recalling long or complicated commands. Entries can be scrolled through in the normal way that one would navigate the BASH history as well (e.g., by using the up and down arrow key) while using the reverse-i-search.

2.6.2 Command shortcuts in the shell: alias and unalias

Alias is a tool I can hardly live without, it grants the user the power to define specific actions for the terminal to execute based on pre-defined commands that you link to an 'alias' (which is more or less like an abbreviation but can in effect be any non-special characters). This gives several capabilities: one of the simplest but most headache-alleviating sometimes is to tell the command line to replace typos with the proper command; another implementation is to pre-define annoyingly cumbersome or complicated commands and run them under an alias. There is excellent coverage of how to use alias all over the internet and I'm sure you'll be able to define a few aliases for yourself immediately but I can think of some pertinent examples which I find to be helpful.

Consider the case of the command ls. Because I'm not an exceptionally skilled typist, I'll frequently commit the typo sl as a replacement for ls which usually throws a monkey wrench into my flow working in the shell. Alias can help! Consider the following terminal transcript:

```
MainUsers-iMac:BASH mainuser$ sl
    -BASH: sl: command not found
MainUsers-iMac:BASH mainuser$ alias sl='ls'
MainUsers-iMac:BASH mainuser$ sl
    Steven_P4_0720_Gonio_AlN_1_Aug_6_Si_Sub.csv
    Steven_P4_0720_Gonio_AlN_1_Aug_6_Si_Sub_UNIX_Format.csv
    UNIX_Steven_P4_0720_Gonio_AlN_1_Aug_6_Si_Sub.csv
    tmp.dat
    x-ray_data_normalized.dat
    x-ray_data_normalizer.sh
MainUsers-iMac:BASH mainuser$ unalias sl
MainUsers-iMac:BASH mainuser$ sl
    -BASH: sl: command not found
```

To recap the terminal session included immediately previously, I tried to issue the command sl to BASH. Because that is not a valid command, BASH rejected it and returned a 'command not found' string. Since sl is (at least for me) a common typo for 'ls' I use alias to create a link between sl and ls by issuing the command sl='ls' (it is important to not include spaces after or before the = sign). Attempting to pass the sl command again after setting it as an alias returns the expected output of ls. Convenient! Finally, for the sake of completeness of the example, you can use unalias to remove the alias you just created. After running unalias sl and trying to pass the sl command again, we find that BASH will return a command not found string.

Alias can be used to issue more complicated commands as well, this may be especially helpful when you have several similar commands in your history and the reverse-i-search can't return your wanted results as easily as just setting a new alias. An alias I find exceptionally useful is when I am attempting to connect to a supercomputer with ssh like the following example (I have truncated some of the output here for the sake of saving space):

```
MainUsers-iMac:BASH mainuser$ alias s='ssh -l uname -Y uname.uname.com'
MainUsers-iMac:BASH mainuser$ s

Login connection to host uname :

Password + OTP:
```

I find this to be immediately practical for time savings 1) in the actual typing or searching for a previously run command, and 2) in eliminating a lot of the hassle with passing typos or mistakes in a command. I'm a big fan of creating large personal dictionaries composed of these sorts of aliases mostly because I like to expend my efforts more on things that cannot be automated instead of grinding away at things that could have been automated by some means. Basically this means that you don't have to be a good typist to be a good typist, all you need is some moderate skills in the terminal! Hooray BASH!

You can use this all sorts of ways and creating a simple BASH program (left as an exercise to the reader at the end) to implement many of your own aliases all at once is a good way of keeping track. Another useful incarnation of this would be to execute programs or scripts or do some operation while you are not in that directory and save yourself the hassle of using many cd commands or writing out a whole path every time you want to run a program or perform an operation that can be automated.

An excerpt from one of my personal alias dictionaries is as follows (there are lots of little tricks within that I very much enjoy using and find to be time-savings):

```
alias sl='ls'        # Correct for spelling errors
alias ks='ls'        # Correct for spelling errors
alias xs='cd'        # Correct for spelling errors
alias alisa='alias'  # Correct for spelling errors
alias ckear='clear'  # Correct for spelling errors
alias whcih='which'  # Correct for spelling errors

alias cd..='cd ..'               # Quick change of directory

alias .2='cd ../../'             # Quick change of directory
alias .3='cd ../../../'          # Quick change of directory
alias .4='cd ../../../../'       # Quick change of directory
alias .5='cd ../../../../../'    # Quick change of directory

alias ..='cd ../'                # Quick change of directory
alias ...='cd ../../'          # Quick change of directory
alias ....='cd ../../../'      # Quick change of directory
alias .....='cd ../../../../'  # Quick change of directory

alias c='clear'                    # Replaces clear with c
alias f='find . |grep '            # Collects user input after the f command and
    searches with grep for any matches
alias lc='wc -l'                   # Count lines in a file
alias lt='du -sh * | sort -h'      # Lists the file sizes of all items in a
    directory in order of size
alias mk='mkdir'                   # Replaces the mkdir command as mk, since I have
    the need for speed
alias nf='ls | wc -l'              # List of number of files within a directory
alias sz='du -sh *'                # Lists the file sizes of all items in a
    directory
alias cco='cd */; cat OSZICAR'     # Enters the only directory and cats out the
    OSZICAR file, useful in test operations
alias cls='clear; ls'              # Replaces clear; cd ..; ls as cls, since I have
    the need for speed
alias lsr='ls -r *'                # List items recursively in a directory
alias rmr='rm -r'                  # Replaces the rm -r command as rmr, since I have
    THE NEED FOR SPEED
alias rnf='ls -r * | wc -l'        # Recursive list of number of files within all
    subdirectories
alias rmev='rm *EV*'               # Deletes all files containing the string EV from
    a directory
alias untar='tar -zxvf'            # Unpack tar files
alias mktar='tar -czvf'            # Pack tar files tar -czvf name-of-archive.tar.gz
    /path/to/directory-or-file
alias vimsh='vim *.sh'             # Runs vim on a shell script (only use if there
    is just a single *.sh file in the directory)
alias calc='cd ~/uname/calc'  # directory location
alias codes='cd ~/uname/codes'  # directory location
alias scripts='cd ~/uname/scripts'  # directory location
```

2.7 Time-based commands: date, cal, sleep, tty-clock

date and cal are mostly straightforward programs, date I find to be immediately useful in a variety of situations because it will print the date and time (or really whatever you want regarding the current time). I find this to be useful when I want to add timestamps to a program to see how long individual sections take to run, or to print time steps when web scraping so its easier to dump into a plotting program later on. cal is more of a convenience sometimes when you want a calendar so I thought I'd add it here, however the basic implementation is all that I'll cover because I think it has limited utility in the scope of this text. Running the date and cal commands in the terminal individually will give you the following output:

```
MainUsers-iMac:BASH mainuser$ date
    Sun Aug 29 18:32:29 PDT 2021
MainUsers-iMac:BASH mainuser$ cal
        August 2021
    Su Mo Tu We Th Fr Sa
     1  2  3  4  5  6  7
     8  9 10 11 12 13 14
    15 16 17 18 19 20 21
    22 23 24 25 26 27 28
```

```
   29  30  31
```

For me, when collecting data especially, I find it useful to print the date in terms of seconds from the start of UNIX time. This eliminates a lot of code clutter because you can just read a simple single number into your plotting program (e.g., the number of seconds since the start of UNIX time) and convert that to the current date. An alternative that I find to be cumbersome is to have the plotting program read a formatted string of numbers like month, day, year, hours, minutes, seconds. One way of printing the UNIX time currently in seconds is by issuing the following command:

```
MainUsers-iMac:BASH mainuser$ date '+%s'
    1630306447

```

The output is the number of seconds since January 1st, 1970 at 00:00:00 UTC. E.g., it has been 1630306447 seconds since January 1st, 1970 00:00:00 UTC at the time of this writing. We will cover more on the use of the date command and various formatting options that you have when invoking it in the scripting section of this text which starts in section 5.

Sleep is a particularly useful time-based command that allows you to control the timing between execution steps in a program e.g., in between individual commands in a set or between steps in a loop. We'll see this used in several examples in shell scripting, particularly in section 5.2, and when running programs or submitting jobs to certain types of queues on a cluster like in section 9. Even though the specific results might be just a bit underwhelming at the current time, we can get a sense of how the sleep command works by running the following example commands that tell BASH to sleep for 1 second, 10 seconds, and 10 minutes respectively:

```
        sleep 1
```

```
        sleep 10
```

```
        sleep 10m
```

As with most things in the shell, the sleep command can be escaped by using the ctrl+c command.

tty-clock is a neat little program mostly for looking cool...and telling the time I guess...but mostly for aesthetics. All tty-clock really does is to make a nice little continuously updating clock in the shell that sort of resembles one of those old-fashioned paper card flip clocks/displays. Because I can't copy verbatim what is output to the terminal by tty-clock, I'll include an approximation after the following command to run tty-clock in my normal way (tty-clock has many options, peruse them at your leisure):

```
sudo apt-get install tty-clock
tty-clock -c -C 1 -t -s
```

The output of tty-clock run this way resembles the following (except for the fact that the real thing, with the options we gave tty-clock, will have red text):

```

               #  ####   #### ####   #### ####
               #  #  # #    #    # #    # #  #
               #  #  #   #### ####   #### #  #
               #  #  # #    #    # #    # #  #
               #  ####   #### ####   #### ####

                           2021-08-24  [AM]
```

2.8 Linux services: service

Service is a very interesting program that can be used to monitor, start, stop, or restart services and daemons in Linux. This can cause significant trouble with your system if you are halting essential services using super used privileges. However sometimes it is essential to know how to manipulate the services ruining on your system. This is perhaps an antiquated example but there were a litany of common problem for some wifi drivers in the Ubuntu 14.04 era. One way of fixing some of the problems, at least temporarily, was to halt and restart the network manager. This could be done using the following command:

```
sudo service network-manager restart
```

The status of all services running on the system can be checked with the following command (basically we're running all of the initialization scripts for all of the services on the system and printing them to the terminal in alpha-order):

```
uname@debian-micro:~$ sudo service --status-all
 [ + ]  apparmor
 [ + ]  chrony
 [ + ]  cron
 [ + ]  dbus
 [ + ]  haveged
 [ - ]  hwclock.sh
 [ + ]  kmod
 [ + ]  networking
 [ + ]  procps
 [ + ]  rsyslog
 [ + ]  sendmail
 [ + ]  ssh
 [ - ]  sudo
 [ + ]  udev
 [ + ]  unattended-upgrades
 [ - ]  x11-common
```

It is worthy of note again that manipulating services in your system can lead to undesired effects if you aren't being careful. The extent of how much this can foul up a system is pretty evident in the following terminal output where Debian returns a 'command not found' when we try to run service with anything but super user privileges:

```
uname@debian-micro:~$ service --help
-BASH: service: command not found
uname@debian-micro:~$ sudo service --help
Usage: service < option > | --status-all | [ service_name [ command | --full-restart
     ] ]
```

In this instance, Debian appears to be deliberately hiding the service command to anyone other than those who know exactly what they're looking for. In the scope of the damage that can be done to a system by misusing the service command and that I'd also agree its not usually needed to be manipulated by the user, that may or may not be justified.

3 Shell Programs and How to Run Them

The differences between shell programs and shell commands can be subtle in some instances so I will not make a great effort here to institute rigorous definitions of each since it would not contribute much to this work. That being said however, I had to make case-by-case decisions on what programs and commands to include in what sections so as to not make any one section too daunting or time-consuming to reference.

In general, I would say that a command references a program and passes options, file names, etc... from the user to the program via the shell. However, programs can be single commands themselves in the shell. For the reasons of that semi-ambiguity, I have made judgement calls of what to call a command and what to call a program in the previous and the current sections. Some shell programs have already been covered in the commands section because I felt that they would be more fittingly placed where they are. In the following subsections, we'll cover some shell programs that I think have great value and personally use regularly.

Running programs in the shell is simple, if you have indeed installed the Ranger file explorer into your shell as shown in the previous section, then enter the following into your terminal to execute the program:

```
ranger
```

Use the arrow keys to navigate between directories and files without the use of the cd command. This program is extremely helpful and more helpful, in many cases, with the exploration of files than a typical graphical user interface shell.

Ranger is just one of an extremely large number of programs which can be installed via the command line on Linux and UNIX systems. Some useful shell programs will be discussed in the remainder of this section.

3.1 System monitors: top, htop, atop, and gotop

htop is a system monitor that seems to have been borne out of another system monitor called top. Top appears to be standard software included with most BASH distributions that I have seen, however htop is not usually standard and needs to be installed with the following command:

```
sudo apt-get install htop
```

The system monitor top (also called a process viewer) will more likely than not be installed on your system currently. htop is, as explained by the developers, "An interactive process viewer for Unix [and Linux]." Atop is another process viewer/monitor that can be more useful for servers and clusters than htop or top in some instances. Install and run htop with the following command (users on Darwin or Fedora-like systems will have to make substitutions for the 'apt-get' command with 'brew' or 'yum' respectively):

```
htop
```

Use the arrow keys to navigate in htop or enter 'q' to quit htop. The program has its own instructions which will not be repeated here, I invite you to use the manual page for htop in order to see its full capabilities.

Top and atop have differing features but, as this author is concerned, htop is the most helpful, re-configurable, and visually appealing.

One excellent use for all of these programs is their ability to kill existing processes running on your computer while giving you a real-time update of such programs system load, memory and CPU percentages and their parent processes.

There are a range of similar system monitors including another called gotop which some people prefer. I generally find that top is the quickest to use on systems that are not your own because htop may be not be installed and you may not have installation privileges on those systems. For my own machines I prefer vastly to use htop, it mostly boils down to increased functionality in that you can kill processes in an almost gui-like way from htop very easily and that the information is presented in a much more pleasing to look at way with nice colors and constant width columns that are more human-readable. Additionally, htop is user-configurable which I find aids me in quickly finding the information I care about and excluding what's superfluous.

3.2 whoami, uname and hostname

About whoami, uname and hostname, there is little to be said. Simply, the program whoami will print the user ID of the account which a user in currently logged into. Uname is slightly more interesting than whoami: upon exploring the manual page, the reader will find that uname will, depending on the option(s) that they select, uname will supply a user with information about the computer and the current user. For instance, uname can tell you the kernel which you are running, the operating system and other pertinent information. The most helpful of all these commands may be the following:

```
uname -a
```

A user will find that all of the pertinent information about the system, in brief terms, has been printed to the terminal.

Hostname has interesting utility inasmuch as it will tell a user their IP address on the sub network to which that person is attached. Consider the following commands:

```
hostname ; hostname -I
```

The first of these two commands will return the name of the user on the system, the second of these commands will return all of the IPv4 addresses attached to the host. The latter can be helpful for connecting two computers on the same network via SSH.

3.3 Network monitoring: bmon, nmap, arp and ping

There exist many network tools for the Linux and UNIX shell however, three of the best are bmon, nmap and arp.

Installing these programs is simple and follows the installation rules as above, listed again here for convenience ('program' in these examples should be substituted for whatever shell program you are intending to install):

```
sudo apt-get install 'program'
```

```
brew install 'program'
```

```
sudo yum install 'program'
```

Specifics of these tools can be found in their respective manual pages but some interesting features will be listed here. We will start with bmon, arguably the simplest of all the programs in this subsection. Run in the terminal the following:

Bmon is (as its name might imply) a bandwidth monitor which gives the user information about receive and transmit (rx and tx) bits on a per-interface basis as well as printing to the terminal a semi-GUI of the network traffic in real time (many features not listed).

```
bmon
```

Nmap is an excellent tool which can be used to determine who is on a network (e.g. the network map, its namesake), what device they are using and what operating system that device is using. I find that nmap is useful in determining how devices are accessing your network and/or if there are devices you don't recognize and may consider to be malicious.

This program will list the used IPv4 addresses being used on the network to which the user is attached, the address in the command can be substituted for any valid IPv4 address. Nmap has excellent utility in network exploration and port scanning but these topics are beyond the scope of this text.

The user can try the following command after installing nmap:

```
nmap -sn 192.168.0.1/24
```

Press on your keyboard the letters 'd' and 'i,' these will bring up detailed network statistics as well as a real-time pseudo-GUI of the network receive and transmit load as a function of time.

Nmap is a simple program to run and the reader is strongly encouraged to read the manual page for nmap and, for that matter, any program that you run. After installing nmap in the normal way, run the following command:

Arp (address resolution protocol) is excellent for determining information about the media access controls of devices attached to a network and will print the sub-net addresses of all the devices running on the network to which you are attached. Additionally, arp can manipulate the IPv4 network cache inasmuch as that it can add, manipulate or delete entries from the network neighbour cache. A simple way to start using arp is to issue the following command (this will hardly scratch the surface of the program's capability):

```
arp -a
```

Ping is a helpful command which issues packets to a known gateway that elicit a response from that gateway, the response will be printed with relevant statistics onto your terminal. To exit this command the use of ctrl+c will be helpful. Consider running the command:

```
ping 8.8.8.8
```

This command will send packets to the 8.8.8.8 Google DNS server and receive back a response. Among its many applications, Ping is particularly helpful when configuring clusters for checking whether nodes are online, and for generally determining whether a connection to a particular server or address is accessible. Conveniently, ping comes pre-installed in most UNIX and Linux based systems.

3.4 Process controls: no hangups and forks

As can be seen in the previous sections, it is infinitely useful to run programs in the terminal. Running programs in the terminal can have drawbacks however; one such case is that (without what will be discussed in this section) the terminal must remain open while the program is running unless special actions are taken. Two possibilities for running programs after the terminal is closed are called no hangups, and forks. Let's take, for example, the case of running PyRoom (one of this author's favorite free text editors).

Run the program with the following (assuming that it is installed on the system):

```
pyroom
```

Now, if you were close the terminal window, PyRoom would close along with it. This is a double-edged sword: if the user is running a program which has a high propensity for crashing or other catastrophic failures like locking up and freezing, it can be helpful to close the terminal in order to halt the program; other times, when a program is running in the terminal and you want to continue using that terminal, a user might find themself out of luck since the terminal session will be occupied with that program.

The workarounds for these problems are called no hangups, and the fork. No hangups is way of running a program which ignores problems (like closing the terminal when a program is running a.k.a hangups). When a program is running with nohangups you will not have access to use your terminal window for anything else before the program is closed. The fork is different; forking a program means that you will run the program you want and get your terminal back immediately (that is you can issue new commands or run new programs in the same terminal); you will see, however that, when you close the terminal, any programs running in it will be closed as well.

Now let's try running PrRoom with no hangups, the fork, and a combination of each. Enter into the terminal the following:

```
nohup pyroom
```

Now try closing the terminal in which you have run the program, you will find that PyRoom will stay open even though the terminal is closed: this is the power of nohup (the command to run with no hangups). Close PyRoom and enter into the terminal the following:

```
pyroom &
```

Try entering a new command into the terminal, you will see that the terminal is available to process new commands; if, however, you closed the terminal, you will see that PyRoom will close with it. This is the power of the fork (the fork is entered into the terminal after a command as the & option).

An excellent usage of both these programs simultaneously is to call them on the same program, this can be done in the following way:

```
nohup pyroom &
```

Running the above command will run PyRoom without hangups and fork the program, immediately giving you back access to the terminal.

3.5 find

Find allows the reader, as is said in its man page, to "...Search for files in a directory hierarchy." Find is a particularly useful program in many instances where analogous operations with the ls program would be cumbersome. A good instance of this is to list all of the files in the working directory which are executable. Refer to the Ownership and Change Mode section (2.4) of this document and create an executable file, or multiple if that's more interesting. Now run the following:

```
find . -maxdepth 1 -perm -a+x -type f
```

The reader will find that the newly created executable(s) as well as any pre-existing in the working directory have been printed to the terminal. Elements of this command are the -maxdepth option which specifies how many sub-directories will be be searched for the argument (1 searches only the working directory), the perm option accepts an argument of what the permissions the files that find is looking for should have (the a+x can be replaced by 111 which is just to say that only a special type of user can execute the file), and the -type f is a cost-based optimizer looking for regular files (can be replaced to search for other file types, see manual).

A practical usage here may be to find log files that have been buried in layers of directories after a failed compilation (which is, humorously, for myself at least, all too common) running a search for log files may look like the following:

```
uname@uname:~> find ~/uname/ -type f -name config.log
~/uname/codes/exciting_smp/external/FoX/config.log
~/uname/codes/exciting_smp/src/libXC/config.log
~/uname/codes/exciting_mpi/external/FoX/config.log
~/uname/codes/exciting_mpi/src/libXC/config.log
~/uname/codes/exciting_serial/external/FoX/config.log
~/uname/codes/exciting_serial/src/libXC/config.log
~/uname/exciting/external/FoX/config.log
~/uname/exciting/src/libXC/config.log
```

3.6 wget and curl

Downloading large numbers of files from online repositories manually can quickly become excessively tedious. Wget is a "GNU utility for downloading network data," as per the program's manual. Wget is one of the most popular programs for downloading information off of FTP servers but it works excellently on HTTP and HTTPS protocols as well. Non-interactive and link-following operation make the program especially helpful so that download processes can be run in the background or while you're away at lunch (for example). Database collectors rejoice, we're going to learn today!

Let's try an example to download an online database of information. To avoid any trouble, I'm going to omit any specifics of website names and instead just refer generally to 'website.' On many database websites, navigating to each page and individually downloading the file you need would be laborious and I certainly don't want to spend all of that time to get the files that I want. Additionally, it is usually useful to be able to access databases offline if possible. In order to try and recursively download a database to your computer, run the following code in your terminal (if wget is not installed then install it the usual way). Wget is also smart enough to make its own directories and sub directories, if all goes well you should be able to browse your favorite databases offline (disk space beware).

```
wget -r http://website
```

Wget should have run and printed a large number of operations to the terminal consisting of what file it is downloading, where it is being downloaded from, where it is being placed on your disk, and a nice status bar of the download progress. The -r tag specifies a recursive download e.g. grab the files from the directory you specify and those related to it.

Wget can also be run without the -r option on a single file with known location, see the following terminal session as an example:

```
uname@uname:~> wget http://exciting.wdfiles.com/local--files/oxygen/exciting.oxygen.
    tar.gz
--2021-08-18 11:50:23--  http://exciting.wdfiles.com/local--files/oxygen/exciting.
    oxygen.tar.gz
Resolving exciting.wdfiles.com (exciting.wdfiles.com)... 107.20.139.170
Connecting to exciting.wdfiles.com (exciting.wdfiles.com)|107.20.139.170|:80...
    connected.
HTTP request sent, awaiting response... 200 OK
Length: 15182668 (14M) [application/x-gzip]
Saving to: 'exciting.oxygen.tar.gz'

exciting.oxygen.tar.gz 100%[=======>]  14.48M  15.8MB/s    in 0.9s

2021-08-18 11:50:24 (15.8 MB/s) - 'exciting.oxygen.tar.gz' saved [15182668/15182668]
```

Another means of downloading files within the shell is the program named curl which has its own costs and benefits compared to wget. While wget is my go-to program when I can get it, it's also not natively available on Mac OS (at least at the time of this writing) whereas curl is. An example of using curl to download a large(ish) text file of the Webster's dictionary from the Project Gutenberg website (for various reasons I find it super useful to have a plain text dictionary available!) is included below:

```
MainUsers-iMac:Desktop mainuser$ curl https://www.gutenberg.org/files/29765/29765-8.
    txt --output websters.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 27.5M  100 27.5M    0     0  2517k      0  0:00:11  0:00:11 --:--:-- 8418k

```

Curl will automatically download the file 29765-8.txt from gutenberg.org and then write it to the directory of your choice: the desktop in my case. I do personally like the overall design of the data monitor (shown above as a terminal output after the file had completed transferring) on curl as well. If you so desire, the output of curl can be silenced with the -s or –silent options.

3.7 vim: a text editor in the shell

Vim is a text editor that has some very nice functionality but can be strange to a user just starting to use it for the first time. It is also very important to recognize that Vim commands and functionality are enormously broad and will not be covered exhaustively here. Coverage of Vim that will be provided in this text will be as much as may be needed to quickly edit and create files on a remote cluster for work with data sets or shell scripts (which will be discussed in detail including and following section 5). First, Vim can be installed with the following:

```
sudo apt-get install vim
```

and run with the following on a file (in this case, a file called README.txt):

```
vim README.txt
```

You will see Vim open in the terminal (if you are on Mac OSX then you may need to have X11 or XQuartz installed for Vim to work properly) and the contents of the README.txt file will be printed to the body of Vim. Editing in Vim is different than some other editors like nano or Microsoft Word for example.

Deleting lines or characters is easy from the main screen of Vim. Deleting a character is as simple as navigating to that character with the arrow keys and pressing delete. Deleting an entire line is possible too by navigating to the line that you want to delete and entering the following:

```
dd
```

Pressing the i key will enter the user into insert mode in Vim, this is generally the most useful for my interests and allows the user to add and delete text and lines easily as they would with a normal text editor. Newer versions of Vim also contain nice features like syntax highlighting and indicating layers of grouping symbols like (), [ ], and {}.

A tricky part (at first but simple when the commands are known) about Vim is saving and/or exiting the program. If you are in a mode of the program, for example the insert mode that we just covered, you can press the escape key to escape that mode. There are lots of options from here but the two most useful may be the following:

To save and quit, enter the following after pressing the escape key (wq stands for write and then quit):

```
:wq
```

To quit without saving, enter the following after pressing the escape key (the ! says, in approximate terms, to Vim 'force this command'):

```
:q!
```

If you would like to entirely delete the contents of a file that you are in the midst of editing with Vim, you can press the escape key and then type the following before hitting the enter or return keys:

```
:1,$d
```

Alvin Alexander in his blog at alvinalexander.com lists the context of the different characters issued to Vim here as the following: the character : starts the 'last line mode'; 1 tells Vim to begin at the first line; ,$ tells Vim to continue the command (which we started at line 1) until the end of the file; and the d command obviously means delete.

3.8 Gnuplot

Gnuplot is a fantastic and full-featured tool for plotting using the terminal. It has all sorts of capabilities and supported formats, and can be controlled to the finest minutia you heart may desire. I make extensive use of Gnuplot in my own research and find that with great ease you can make plots from data files that look much nicer than those created by some other programs like MATLAB (which unfortunately could stand to see some improvement in the aesthetics of its plotting engine which is otherwise extremely full-featured). In fact, I make extensive use of Gnuplot in section 5.3 of this text where I demonstrate how to make a web scraping engine all with shell script and use Gnuplot to create a plot readable headless in the terminal! With Gnuplot, you don't even have to write the plot to a graphics file in order to visualize it, you can tell Gnuplot to output the plot as symbols directly to the terminal! I find that feature to be infinitely useful, especially when using remote systems like super computers.

Gnuplot can be installed easily on Linux systems using the following command:

```
sudo apt-get install gnuplot
```

Or on Mac OS by using the following command:

```
brew install gnuplot
```

Running Gnuplot is also simple and can be done in the following way with the following results:

```
MainUsers-iMac:BASH Programs mainuser$ gnuplot

  G N U P L O T
  Version 5.2 patchlevel 2    last modified 2017-11-15

  Copyright (C) 1986-1993, 1998, 2004, 2007-2017
  Thomas Williams, Colin Kelley and many others

  gnuplot home:     http://www.gnuplot.info
  faq, bugs, etc:   type "help FAQ"
  immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'unknown'
gnuplot>
```

Since Gnuplot is all command line tools for creating graphics files, it can be a little bit intimidating to use at first glance. However, I believe that, after just a few minutes working with the program, it is intuitive, quick, and easy in most cases to get running at a high level. I most frequently use Gnuplot for plotting data from large .csv or .dat formatted files and have never found it to hang up with issues like line-ending characters that we discuss in sections 5.7 and 3.13.

In the following examples, I will show use cases of my own with example scripts for the use of Gnuplot, as well as some of its features and workarounds for some of its quirks.

To get started, we can create the following data file that I will name data.dat for simplicity. The file is just integers 1-12 as well as their products with 2 and their squares. You can create the same data file using the following script (execute the script by copying it into a text editor, giving it a name like data_file.dat, executing chmod +x data_file.dat, and then ./data_file.dat):

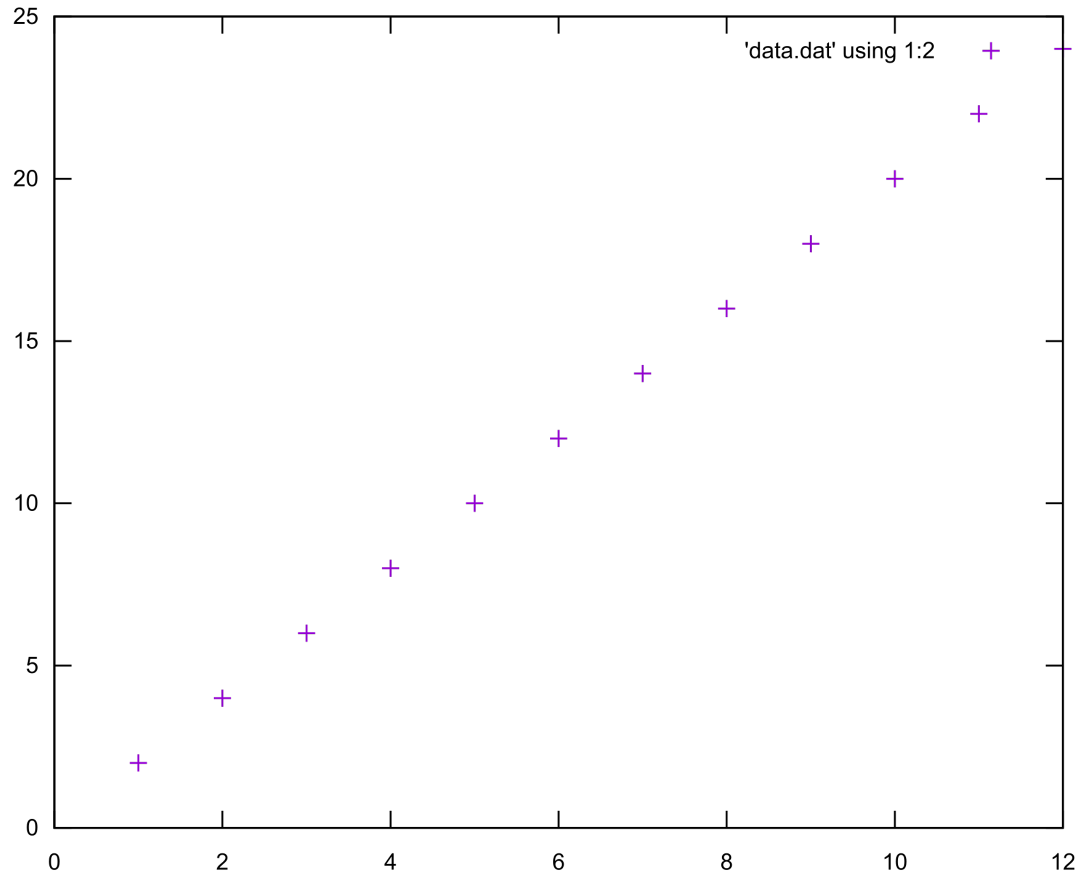
```
#!/bin/bash
cat > data.dat << EOF
1 2 1
2 4 4
3 6 9
4 8 16
5 10 25
6 12 36
7 14 49
8 16 64
9 18 81
10 20 100
11 22 121
12 24 144
EOF
```

Now that you have a data file that can be used with Gnuplot, we can begin to work on plotting the data and creating figures. Gnuplot can create graphics with a range of different formats using different terminals within Gnuplot that are set by the user. I prefer to have .svg files for all of my plots because they can be recolored, re-scaled, and retouched simply after their creation with several programs including Inkscape. For this reason I will discuss mostly the svg-enhanced terminal in this section but I will also cover the "dumb" terminal, which outputs plots directly as text into the terminal (this functionality is used extensively in section 5.3).

To make a simple plot of the data file we just created, we can run the following commands in Gnuplot (assuming that you are located in the same directory as the data.dat file you just created):

```
gnuplot> set terminal svg enhanced

    Terminal type is now 'svg'
    Options are 'size 600,480 fixed enhanced font 'Arial,12' butt dashlength 1.0 '
gnuplot> set out 'data.svg'
gnuplot> plot 'data.dat' using 1:2
gnuplot>
```

What we did was to first define for Gnuplot what file format we want using the set terminal command. Next, we set an output file called data.svg. Finally, we plot the data using the first and second columns in the data file (which are passed to Gnuplot as 1:2). These Gnuplot commands will create the following plot as a salable vector graphics (svg) image:



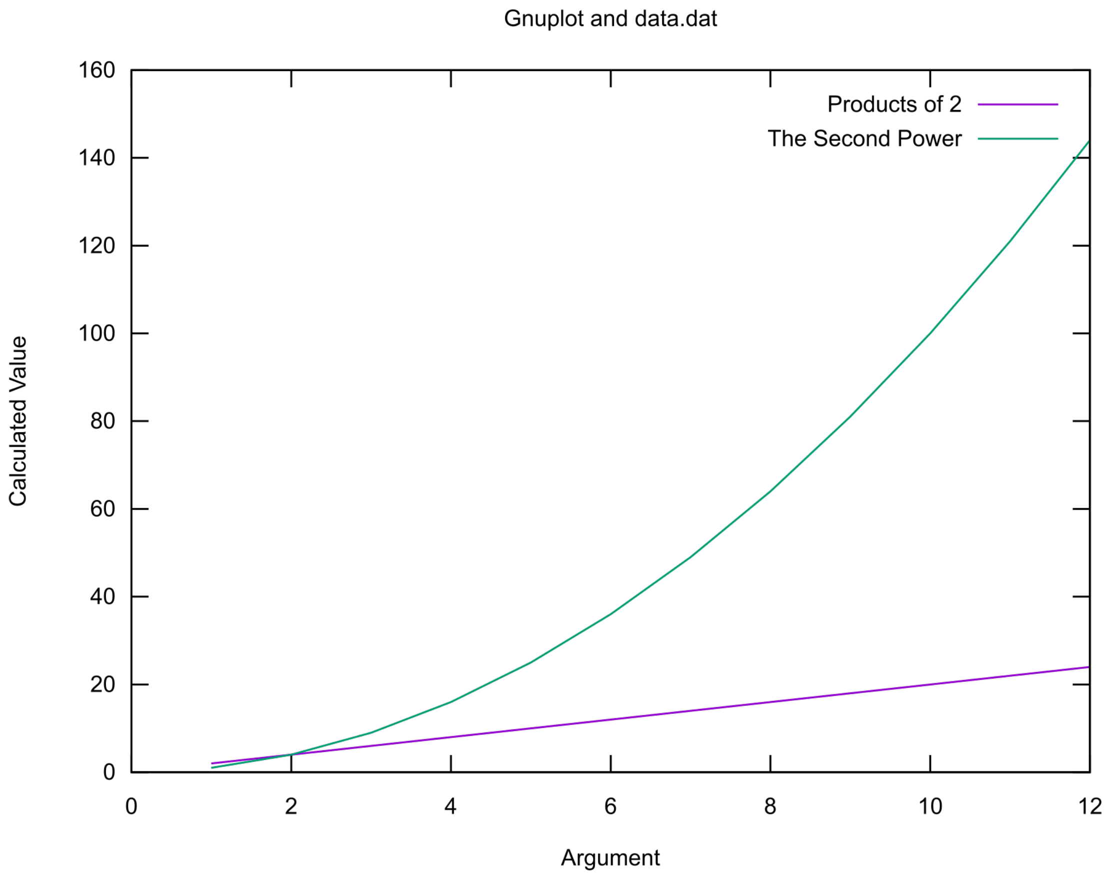
Some of the syntax in Gnuplot is able to be abbreviated. These abbreviations can save a lot of time for the user and I recommend their use for the sake of brevity. Some of these commands and their abbreviations are tabulated in the following table:

line(s)	l
line color	lc
line width	lw
plot	p
quit	q
title	t
using	u
with	w
xlabel	xl
xrange	xr
ylabel	yl
yrange	yr

Gnuplot is excellent at handling multiple sets of data at the same time. If we want to plot all of the data points in the data.dat file using abbreviations, and some extra, fancy options as well, we can do so with the following commands (please refer to the above table of abbreviations for the long names of equivalent commands in Gnuplot):

```
gnuplot> set terminal svg enhanced

    Terminal type is now 'svg'
    Options are 'size 600,480 fixed enhanced font 'Arial,12' butt dashlength 1.0 '
gnuplot> set xr [0:12]
gnuplot> set yr [0:160]
gnuplot> set title 'Gnuplot and data.dat'
gnuplot> set yl 'Calculated Value'
gnuplot> set xl 'Argument'
gnuplot> set out 'data2.svg'; p 'data.dat' u 1:2 t 'Products of 2' w l, 'data.dat' u
     1:3 t 'The Second Power' w l
gnuplot>
```

This time, we use the first and second columns as the first data set, and then the first and third columns as the second data set. Running the above commands will create the following plot:

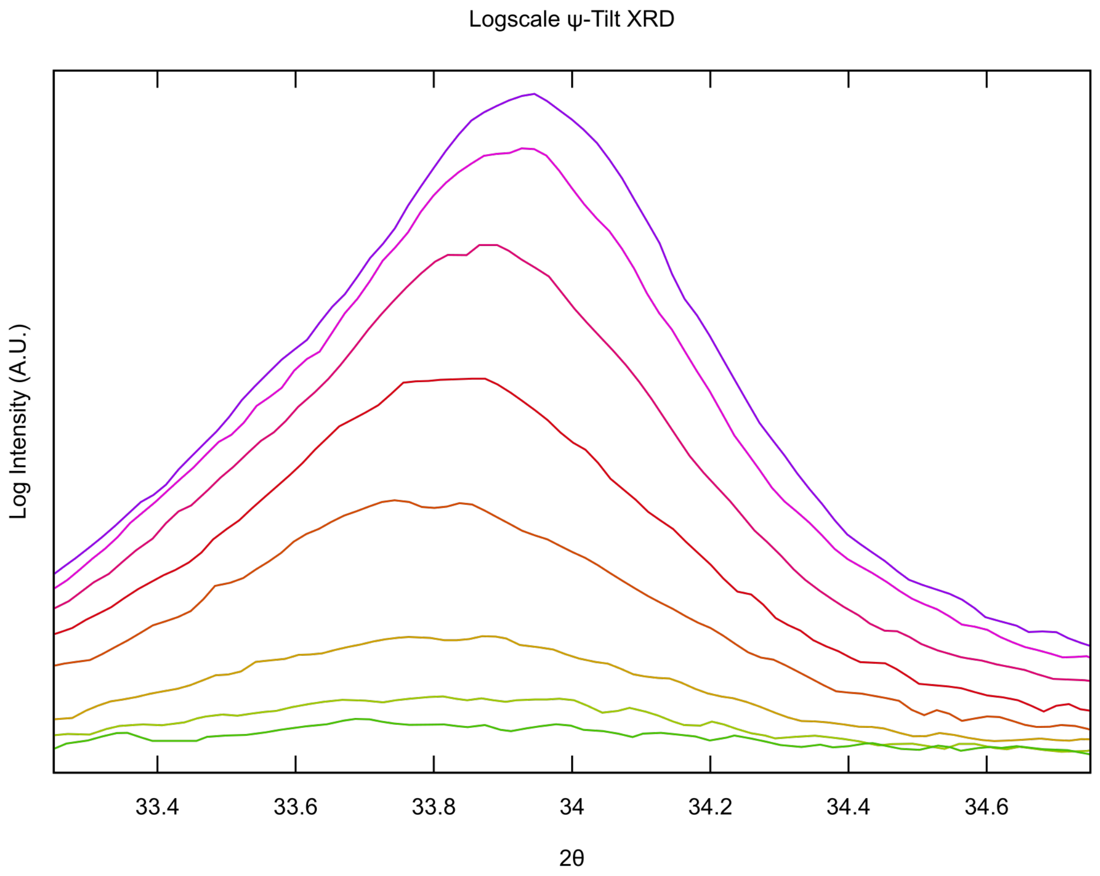


Gnuplot has all sorts of further capabilities, the full scope of which is beyond this text. However, there are many other Gnuplot scripts as well as outputs that I would like to present. For the sake of brevity though I will keep the examples brief and attempt to include as many useful options in the scripts as reasonable.

One case where I use Gnuplot for my own research is in preparing figures for journal publications. Usually I like to have some color and clear labeling in these plots since they will be on record with a journal. One such example is with X-ray diffraction data with multiple psi-angles that I include in a paper [1] (reproduced here with permission). I use Gnuplot to plot the data, as well as to convert the y-values to log scale, and to color the plots so that they look attractive and of publication quality. Because the input data file is long, I will not include it here but I will include the script that generates the plot and the plot as well.

```
set terminal svg enhanced
set xrange [33.25:34.75]; set yrange [5.4:9.5]
set title 'Logscale psi-Tilt XRD'
set xlabel '2_theta'; set ylabel 'Log Intensity (A.U.)'
unset ytics
set border
```

```
unset key
set datafile separator ","

set out 'PsiTilt_XRD.svg'
p 'PsiTilt_XRD.csv' u 1:(log($2)) t '0' lc rgb '#8F00E5' w l,      'PsiTilt_XRD.csv'
    u 3:(log($4)) t '1' lc rgb '#DF00D2' w l,     'PsiTilt_XRD.csv' u 5:(log($6)) t
   '2' lc rgb '#DA006F' w l,     'PsiTilt_XRD.csv' u 7:(log($8)) t '3' lc rgb '#
   D40011' w l,     'PsiTilt_XRD.csv' u 9:(log($10)) t '4' lc rgb '#CF4700' w l,    '
   PsiTilt_XRD.csv' u 11:(log($12)) t '6' lc rgb '#C99C00' w l,   'PsiTilt_XRD.csv'
   u 13:(log($14)) t '8' lc rgb '#9CC400' w l,   'PsiTilt_XRD.csv' u 15:(log($16)) t
    '10' lc rgb '#46BF00' w l
```



In this plot, I use several useful functionalities of Gnuplot. First, I set the ranges in the x- and y-directions using the xrange and yrange commands. Next, I set the title of the plot as well as its x- and y-labels. Next I remove all tic markings from the y-axis because the data is in log of arbitrary intensity and used to show drop-off of a relative value. After that I set a border around the plot, unset the plot's key, and tell Gnuplot that commas are used as the data file separator.

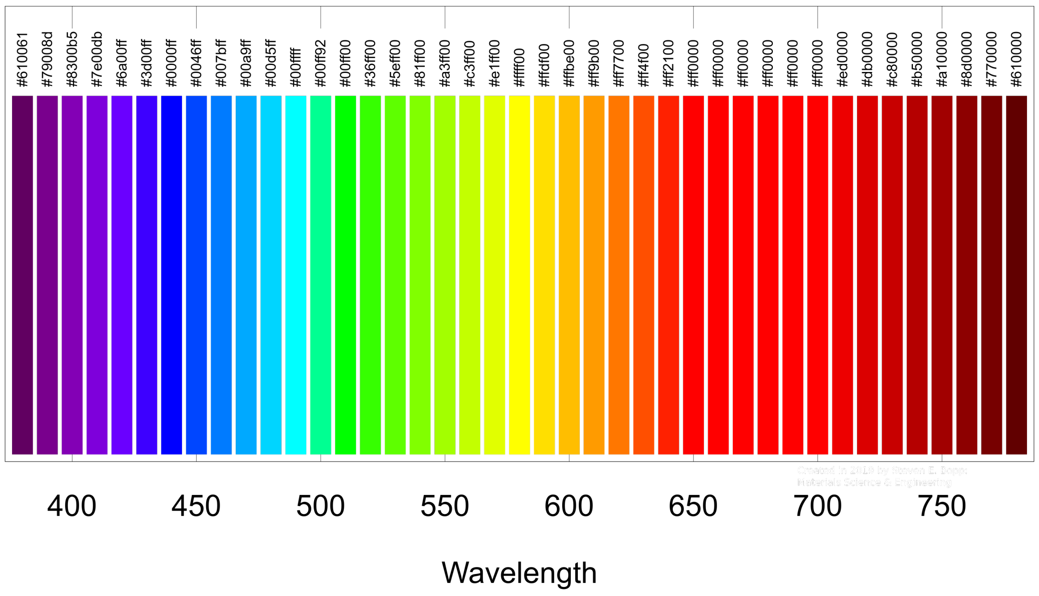
Data file separators are important and can throw errors in Gnuplot that are concerned with the readability of the data set. If you have spaces separating all of your data value columns in the data set, then Gnuplot will not need to be told specifically what the data file separator is since that seems to be its default setting. In the case of the common comma separated values (.csv) file format, values are obviously not separated by spaces and therefore you need to tell Gnuplot specifically that commas are used. Remembering this can alleviate a lot of headaches down the road.

Finally, I set the name of output file as a title relevant to what I'm working on, and then plot the data. Plotting the data has two interesting commands that we will explore. First, I use the mathematical capability of Gnuplot to calculate the log of the data before I plot it, this is done using the second column in each data set which is the intensity of X-rays that are collected by a detector. Gnuplot stores this data series as a variable and we cal call that variable by placing a $ sign before the column number and enclosing the entire calculation in parenthesis (variables will be covered in greater depth in section 4.1). This makes for a convenient means of re-scaling data on the fly. Finally, I also set the line color that I want by giving Gnuplot a hexadecimal color value to assign to each of the lines individually.

In the following, I include a chart where the user can reference the number of curves that they want to plot with different colors and then copy the numbers of 'evenly-spaced' hex colors from the chart (one of my own creation that I find frequently useful for all matter of plotting operations).

3 Colors	4 Colors	5 Colors	6 Colors	7 Colors	8 Colors	9 Colors	10 Colors	11 Colors	12 Colors	13 Colors	14 Colors	15 Colors
#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5	#8F00E5
#D21B00	#D8004F	#DB0088	#DD00AA	#DE00C1	#DF00D2	#E000DF	#D800E0	#D100E1	#CB00E1	#C600E1	#C200E2	#BE00E2
#46BF00	#CB8000	#D21B00	#D50024	#D8004F	#DA006F	#DB0088	#DC009B	#DD00AA	#DE00B7	#DE00C1	#DF00CA	#DF00D2
	#46BF00	#C8B000	#CE5900	#D21B00	#D40011	#D60034	#D8004F	#D90066	#DA0078	#DB0088	#DC0095	#DC00A0
		#46BF00	#C0C600	#CB8000	#CF4700	#D21B00	#D40007	#D50024	#D7003B	#D8004F	#D90061	#DA006F
			#46BF00	#ABC500	#C99C00	#CD6800	#CF3D00	#D21B00	#D30001	#D50019	#D6002E	#D70040
				#46BF00	#9CC400	#C8B000	#CB8000	#CE5900	#D03700	#D21B00	#D30300	#D40011
					#46BF00	#91C300	#C7C000	#CA9400	#CC6E00	#CE4E00	#D03300	#D21B00
						#46BF00	#88C300	#C0C600	#C9A300	#CB8000	#CD6200	#CF4700
							#46BF00	#81C200	#B4C500	#C8B000	#CA8F00	#CC7200
								#46BF00	#7CC200	#ABC500	#C7BB00	#C99C00
									#46BF00	#77C200	#A3C400	#C7C500
										#46BF00	#73C100	#9CC400
											#46BF00	#70C100
												#46BF00

Another example of using Gnuplot is to make the following figure that compares wavelength of light to an approximately matching hex color for the visible spectral range. I find this useful when I am trying to understand more intuitively a color of light when someone relays to me the wavelength of that light. This example relies heavily on the arrow and label commands which are Gnuplot built-ins. I use arrows without heads and with a heavy line width (lw) of a certain length to create 'pillars' that all have a certain hex color assigned to them based on a series of labels that are all rotated 90 degrees so that they are aligned with the color 'pillars'. Please see the figure below:



To replicate this figure, begin by creating the following text file with the attached script and running it in the normal way by pasting it into a shell script extended file and changing its mode so that it is executable with chmod+x:

```
#!/bin/bash
cat > data.dat << EOF
0,0
0.01,0.01
EOF
```

The Gnuplot script is slightly different than those that we have spoken about before because we are not actually plotting any data here. If you looked at the data file, you will have seen that there are only two points and they actually lie outside of the box that we are plotting. This plot actually only uses the arrow and label functionality and we just include that data file and two points for the program to plot just to keep Gnuplot happy. Plotting this way allows you to create data graphics with Gnuplot outside of its regular data plotting functionality.

```
set terminal svg enhanced size 3840,2160 font 'Arial,100'
set xrange [373:787]
set yrange [-0.02:1.25]
unset ytics
set xlabel 'Wavelength'
set datafile separator ","
set key below

set arrow 1 from 380,0 to 380,1 nohead lw 40 lc rgb '#610061'
set arrow 2 from 390,0 to 390,1 nohead lw 40 lc rgb '#79008d'
set arrow 3 from 400,0 to 400,1 nohead lw 40 lc rgb '#8300b5'
set arrow 4 from 410,0 to 410,1 nohead lw 40 lc rgb '#7e00db'
set arrow 5 from 420,0 to 420,1 nohead lw 40 lc rgb '#6a00ff'
set arrow 6 from 430,0 to 430,1 nohead lw 40 lc rgb '#3d00ff'
set arrow 7 from 440,0 to 440,1 nohead lw 40 lc rgb '#0000ff'
set arrow 8 from 450,0 to 450,1 nohead lw 40 lc rgb '#0046ff'
set arrow 9 from 460,0 to 460,1 nohead lw 40 lc rgb '#007bff'
set arrow 10 from 470,0 to 470,1 nohead lw 40 lc rgb '#00a9ff'
set arrow 11 from 480,0 to 480,1 nohead lw 40 lc rgb '#00d5ff'
set arrow 12 from 490,0 to 490,1 nohead lw 40 lc rgb '#00ffff'
set arrow 13 from 500,0 to 500,1 nohead lw 40 lc rgb '#00ff92'
set arrow 14 from 510,0 to 510,1 nohead lw 40 lc rgb '#00ff00'
set arrow 15 from 520,0 to 520,1 nohead lw 40 lc rgb '#36ff00'
set arrow 16 from 530,0 to 530,1 nohead lw 40 lc rgb '#5eff00'
set arrow 17 from 540,0 to 540,1 nohead lw 40 lc rgb '#81ff00'
set arrow 18 from 550,0 to 550,1 nohead lw 40 lc rgb '#a3ff00'
set arrow 19 from 560,0 to 560,1 nohead lw 40 lc rgb '#c3ff00'
set arrow 20 from 570,0 to 570,1 nohead lw 40 lc rgb '#e1ff00'
set arrow 21 from 580,0 to 580,1 nohead lw 40 lc rgb '#ffff00'
set arrow 22 from 590,0 to 590,1 nohead lw 40 lc rgb '#ffdf00'
set arrow 23 from 600,0 to 600,1 nohead lw 40 lc rgb '#ffbe00'
set arrow 24 from 610,0 to 610,1 nohead lw 40 lc rgb '#ff9b00'
set arrow 25 from 620,0 to 620,1 nohead lw 40 lc rgb '#ff7700'
set arrow 26 from 630,0 to 630,1 nohead lw 40 lc rgb '#ff4f00'
set arrow 27 from 640,0 to 640,1 nohead lw 40 lc rgb '#ff2100'
set arrow 28 from 650,0 to 650,1 nohead lw 40 lc rgb '#ff0000'
set arrow 29 from 660,0 to 660,1 nohead lw 40 lc rgb '#ff0000'
set arrow 30 from 670,0 to 670,1 nohead lw 40 lc rgb '#ff0000'
set arrow 31 from 680,0 to 680,1 nohead lw 40 lc rgb '#ff0000'
set arrow 32 from 690,0 to 690,1 nohead lw 40 lc rgb '#ff0000'
set arrow 33 from 700,0 to 700,1 nohead lw 40 lc rgb '#ff0000'
set arrow 34 from 710,0 to 710,1 nohead lw 40 lc rgb '#ed0000'
set arrow 35 from 720,0 to 720,1 nohead lw 40 lc rgb '#db0000'
set arrow 36 from 730,0 to 730,1 nohead lw 40 lc rgb '#c80000'
set arrow 37 from 740,0 to 740,1 nohead lw 40 lc rgb '#b50000'
set arrow 38 from 750,0 to 750,1 nohead lw 40 lc rgb '#a10000'
set arrow 39 from 760,0 to 760,1 nohead lw 40 lc rgb '#8d0000'
set arrow 40 from 770,0 to 770,1 nohead lw 40 lc rgb '#770000'
set arrow 41 from 780,0 to 780,1 nohead lw 40 lc rgb '#610000'

set label '#610061' left at 380, 1.025 font "arial,40" rotate by 90
set label '#79008d' left at 390, 1.025 font "arial,40" rotate by 90
set label '#8300b5' left at 400, 1.025 font "arial,40" rotate by 90
set label '#7e00db' left at 410, 1.025 font "arial,40" rotate by 90
set label '#6a00ff' left at 420, 1.025 font "arial,40" rotate by 90
set label '#3d00ff' left at 430, 1.025 font "arial,40" rotate by 90
set label '#0000ff' left at 440, 1.025 font "arial,40" rotate by 90
set label '#0046ff' left at 450, 1.025 font "arial,40" rotate by 90
set label '#007bff' left at 460, 1.025 font "arial,40" rotate by 90
set label '#00a9ff' left at 470, 1.025 font "arial,40" rotate by 90
set label '#00d5ff' left at 480, 1.025 font "arial,40" rotate by 90
set label '#00ffff' left at 490, 1.025 font "arial,40" rotate by 90
set label '#00ff92' left at 500, 1.025 font "arial,40" rotate by 90
set label '#00ff00' left at 510, 1.025 font "arial,40" rotate by 90
set label '#36ff00' left at 520, 1.025 font "arial,40" rotate by 90
set label '#5eff00' left at 530, 1.025 font "arial,40" rotate by 90
set label '#81ff00' left at 540, 1.025 font "arial,40" rotate by 90
set label '#a3ff00' left at 550, 1.025 font "arial,40" rotate by 90
set label '#c3ff00' left at 560, 1.025 font "arial,40" rotate by 90
set label '#e1ff00' left at 570, 1.025 font "arial,40" rotate by 90
set label '#ffff00' left at 580, 1.025 font "arial,40" rotate by 90
set label '#ffdf00' left at 590, 1.025 font "arial,40" rotate by 90
set label '#ffbe00' left at 600, 1.025 font "arial,40" rotate by 90
set label '#ff9b00' left at 610, 1.025 font "arial,40" rotate by 90
set label '#ff7700' left at 620, 1.025 font "arial,40" rotate by 90
set label '#ff4f00' left at 630, 1.025 font "arial,40" rotate by 90
set label '#ff2100' left at 640, 1.025 font "arial,40" rotate by 90
set label '#ff0000' left at 650, 1.025 font "arial,40" rotate by 90
set label '#ff0000' left at 660, 1.025 font "arial,40" rotate by 90
set label '#ff0000' left at 670, 1.025 font "arial,40" rotate by 90
set label '#ff0000' left at 680, 1.025 font "arial,40" rotate by 90
set label '#ff0000' left at 690, 1.025 font "arial,40" rotate by 90
set label '#ff0000' left at 700, 1.025 font "arial,40" rotate by 90
set label '#ed0000' left at 710, 1.025 font "arial,40" rotate by 90
set label '#db0000' left at 720, 1.025 font "arial,40" rotate by 90
set label '#c80000' left at 730, 1.025 font "arial,40" rotate by 90
set label '#b50000' left at 740, 1.025 font "arial,40" rotate by 90
set label '#a10000' left at 750, 1.025 font "arial,40" rotate by 90
set label '#8d0000' left at 760, 1.025 font "arial,40" rotate by 90
set label '#770000' left at 770, 1.025 font "arial,40" rotate by 90
set label '#610000' left at 780, 1.025 font "arial,40" rotate by 90

set out 'colors.svg'; p 'data.csv' u 1:2
```

Other terminals may be used with Gnuplot as well, some of the most popular are the .png terminal and the .ps terminal. Additionally, it is very important to note again that data file separators are user-defined for Gnuplot in all cases except for the single space data file separator which is its default. Data file separators can be mostly anything that you'd like within reason. For example, if you had a .csv file where the data file separators (as the name .csv or 'comma separated values' would imply), then you could add the following command to the beginning of your Gnuplot script to change that setting:

```
set datafile separator ","
```

3.9 lynx

lynx is a neat, totally text-based web browser that can be used in the terminal and it can be installed with the following command:

```
sudo apt-get install lynx
```

Running lynx on its own can be kind of tedious, especially compared to a modern browser but can be done for kicks here and there. One excellent functionality however is that lynx has the ability to dump the nicely formatted contents of a web page into a local file. This will become very important in section 5.3 on web scraping. We can try for ourselves dumping a web page with the following command to dump a the contents of a weather report to a local file:

```
lynx --dump https://hpwren.ucsd.edu/Sensors/SDSC/ > lynx_dump.txt
```

Here we use a special character > which is a redirect (we'll cover these in section 4.1 but for now it just takes the output of a command and dumps it into a text file). An example output of this is the following (where some extra text at the end has been trimmed for brevity using another command called sed which we will explore briefly in section 5.3 where we make a web scraper).

```
user@debian-micro:~$ cat lynx_dump.txt
REFRESH(61 sec): [1]https://hpwren.ucsd.edu/Sensors/SDSC/
HPWREN multicast-based weather station data display
20210818 15:00:56 - UCSD San Diego Supercomputer Center: 32.88N 117.24W
400'
This sensor is a Davis met station
graphs are since midnight two days ago
[2]Outside air temperature   23.6  Celsius        74.4  Fahrenheit
[3]Inside air temperature    25.7  Celsius        78.2  Fahrenheit
[4]Outside relative humidity 76    percent
[5]Inside relative humidity  54    percent
[6]Wind direction            227   degrees
[7]Wind speed                2.2   meter/second   5     miles/hour
[8]10min wind speed          1.8   meter/second   4     miles/hour
[9]Air pressure              1006  millibar
[10]Solar radiation          384   watts/(meter^2)
[11]UV                       22    UV Index/10
[12]Rain rate                0     clicks/hour
[13]Disclaimer
```

3.10 openmpi

Serial computations are calculations run on a single computing unit. Running large or 'expensive' calculations, especially with programs that require vast computing resources like VASP and BerkeleyGW in serial may not be tractable due to time constraints. These computations can be run on many computing units at a time with a parallelization software. OpenMPI is a parallelization program for dividing a lengthy calculation between many computing units in order to save time on a calculation. I will not get into depth with how to run this program, however I include the following commands for the installation of OpenMPI packages which have worked for me in the past when compiling Quantum Espresso (which will be discussed at length in section 6) on my own personal computers.

```
sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev
```

3.11 ssh and scp

SSH (secure shell) and SCP (secure copy) are built-in programs to BASH. SSH allows the user to control a remote system via the terminal and SCP allows a user to securely copy a file from a remote machine to their local machine or vice versa. If you are very interested in SCP and its protocols, you may also be interested in searching for how ssh tunnels work but the nuances of these topics are beyond the scope of this text.

Using a supercomputer, almost all of the time you will have to use SSH in order to communicate with the machine. Some newer interfaces actually allow for access to terminals through an internet browser which is wildly convenient but we will continue this example with SSH because many systems still require that sort of connection. One such instance of how you can begin talking to a remote system is with the following command (specific to some supercomputer systems that I commonly use for my own research.:

```
ssh -l uname -Y uname.uname.com
```

An example output of this command (again specific to the supercomputer systems that I commonly use) is as follows:

```
Last login: Sun Aug 22 20:40:32 on ttys001
MainUsers-iMac:~ mainuser$ ssh -l uname -Y uname.uname.com

Login connection to host uname :

Password + OTP:
```

An SSH session can be ended at any time by killing the terminal that the session is running inside of. Alternatively, a more convenient means of ending an SSH session is to press the enter or return key and then type the following into the keyboard before pressing the enter or return key again:

```
~.
```

Issuing tilde and period to the terminal and pressing the return or enter keys will end the SSH session and return to the terminal that the connection to the server has closed.

3.12 bc an arbitrary precision BASH calculator

Unfortunately, at least in my opinion, BASH does not have extensive built-in mathematical functionality. One way to work around this is to use shell programs that you can pipe variables and mathematical operations into (we will discuss more about the idea of a 'pipe' in section 4.2). One such shell program is called bc. From its manual page, "bc is a language that supports arbitrary precision numbers with interactive execution of statements."

What this means, more or less, is that you can select the number of significant figures that you want to use in your calculation; this is done using the scale option in bc. This is extremely handy in many situations like when you are selecting the accuracy you desire for a certain calculation, formatting significant figures to ease work down the line, or by rounding calculations to the nearest integer, tenth of a decimal place, or wherever your heart so desires.

bc can be run in the terminal as its own standalone program, however there are such a large number of useful calculators beyond bc that this is not my normal use case. Where bc shines however is within shell scripts as a command where you want to run quick calculations, especially on variables within the script. bc can be run from the terminal with the following command and mathematical calculations can be run within bc:

```
bc
```

Please see the following which is a terminal log of several commands run using bc to get a feel for how to use the program and its functionality. The default scale set for bc is 1, this can deter first time users because they might ask bc to calculate the square root of 2 for example and they'll see that bc gives them an answer of 1. You can control the scale of the calculation by setting scale=n where n is the number of significant figures you want to calculate. You can escape from bc by typing quit into the terminal.

```
MainUsers-iMac:~ mainuser$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
1+2; sqrt(2); 2^8
3
1
256
scale=0; 3+5; 3-5; 3*5; 3/5;
8
-2
15
0
scale=2; 3+5; 3-5; 3*5; 3/5;
8
-2
15
.60
scale=0; sqrt(2); scale=1; sqrt(2); scale=2; sqrt(2);
1
1.4
1.41
```

bc can also perform a number of operations on variables, one I think is particularly useful is the increment commands that are demonstrated here:

```
MainUsers-iMac:~ mainuser$ var="1"
MainUsers-iMac:~ mainuser$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
++ var
1
++ var
2
++ var
3
quit
MainUsers-iMac:~ mainuser$ echo $var
1
MainUsers-iMac:~ mainuser$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
var ++
0
var ++
1
var ++
2
var ++
3
quit
MainUsers-iMac:~ mainuser$ echo $var
1
```

For me, a particular use case of bc is in the creation and management of input files for DFT and other calculation engines like VASP and Exciting. VASP prefers to have an NPAR option in its INCAR file (we will discuss VASP more substantively in section 9) that is approximately the square root of the number of cores that you are calculating with when you are running a parallel computation; this can be quickly done with bc as is shown below. I automate this for my scripts because I frequently change the number of nodes that I use for calculations and would otherwise forget to change this parameter every time I change my scripts. Automation here saves me a lot of effort and spares me sometimes re-running calculations and wasting time with improperly defined parameters. I also find bc very useful for the conversion of units like angstroms into atomic units like Bohr radii, this I do automatically by extracting values from a VASP POSCAR file as variables using awk and then converting them with a conversion factor using bc. These conversions are immediately useful in programs like Exciting which will be discussed at greater length in section 7

The following quick script creates three variables and an input file for VASP, I use this as part of my setup when I run VASP calculations in parallel on remote clusters. In the script, the first two variables are user-defined and are a small name for the calculation to use as well as a number of job nodes. The job node variable is used in SLURM scripts for queuing and running jobs as well. Next we use bc to calculate the nearest integer to the square root of the number of job nodes which is an optimization parameter for the INCAR file. This is done by creating NPAR as a variable that is defined as the echo of the square root of the Job_Nodes variable into bc with a scale set to 0 (which will return the nearest integer to the square root).

```
System_Name="FCC_Al" # Give a calculation title for VASP
Job_Nodes="64"       # Give the number calculation nodes

NPAR=$(echo "scale=0;sqrt($Job_Nodes)" | bc)
# Calculate NPAR for INCAR file [~sqrt(job cores)];
# scale=0 sets bc to round to nearest integer for VASP


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:- Create INCAR File for VASP :-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

cat > INCAR << EOF
# General Setup
  System = ${System_Name} # Calculation Title
  PREC   = NORMAL         # Options: Normal, Medium, High, Low

  ISMEAR = -5
  IBRION = 2
  ISIF   = 3
  LWAVE  = .FALSE.
  LCHARG = .FALSE.
  ENCUT  = 250.00

# Parallelization
  NPAR     = ${NPAR}      # approx. SQRT(number of cores)
EOF

echo " Writing input file INCAR..."
echo "  done"
```

Of course you do not have to be relegated to a certain level of precision or another, that is one of the best features of bc in my opinion. You can define what ever level of precision you want as long as it is within reason. Above, we saw that using the scaling set to zero was convenient for a specific case where we wanted bc to return the nearest integer to the square root of a number of nodes but we can run all sorts of calculations with all sorts of levels of precision:

```
MainUsers-iMac:~ mainuser$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
sqrt(64); sqrt(32); sqrt(16); sqrt(8); sqrt(4)
8
5
4
2
2
scale=10;sqrt(64); sqrt(32); sqrt(16); sqrt(8); sqrt(4)
8.0000000000
5.6568542494
4.0000000000
2.8284271247
2.0000000000
scale=50;sqrt(64); sqrt(32); sqrt(16); sqrt(8); sqrt(4)
8.00000000000000000000000000000000000000000000000000
5.65685424949238019520675489683879231427868750150779
4.00000000000000000000000000000000000000000000000000
2.82842712474619009760337744841939615713934375075389
2.00000000000000000000000000000000000000000000000000
```

3.13 dos2unix, converting old dos files to UNIX format

We will talk slightly more about misbehavior that can come from attempting to perform shell operations on files that are formatted for different systems than UNIX in the scripting section that covers data manipulation and normalization in section 5.7 on page 95 of this text but various parameters can interfere with improper operation of BASH programs when files are not formatted properly. A common incarnation of this problem is when one tries to use BASH in Mac OS, Linux, or UNIX on an old DOS-formatted file. These old DOS-formatted files are in fact very common because many of the research instruments and machines that you will find in universities are non-networked old Windows XP systems (or perhaps even older *gasp!*). Of particular annoyance is the carriage return character in DOS which is very different from the UNIX line ending character. Some programs, awk is an example of these, will not like to operate on files that have the carriage return character and will say that there is an illegal character and then escape out of the operation that you have wanted them to perform. An example of this is the following error:

```
(standard_in) 1: illegal character: ^M
```

One way to work around this is to use a program to convert the format of the old dos-formatted file into the UNIX format. This can be done actually with the program tr (translate) as will be seen in section 5.7 but it can also be done with other programs that are purpose-built. One such program is known as dos2unix and can be installed with the following command:

```
sudo apt install dos2unix
```

This can easily be run on a DOS-formatted file (I find that dos2unix works easiest on Linux systems) with the following where dos2unix will create a new file that is of the UNIX format:

```
ubuntu-budgie@ubuntu-budgie:~/Desktop$ dos2unix DOS_file.csv
dos2unix converting file DOS_file.csv to Unix format...
```

Deeper characteristics of these individual formatting standards are beyond the scope of this text but suffice it to say that there are several formats like these and that I find the most universal of these to be the UNIX format. If you are confronted with errors that report an illegal character, consider checking whether the line endings and formatting are what BASH and/or whatever program you are running is expecting.

It is worthy of note that, if you are attempting to have an old DOS system read a file which is formatted for UNIX or another standard, dos2unix can also be run the opposite direction. That is to say that dos2unix can also convert UNIX formatted files to the DOS format by running the unix2dos command.

4 Logical Operations: Variables, Redirects, Pipes, While, For, and If

Redirects and pipes, as well as how they interact with various types of loops underpin the incredible modularity and versatility of the shell, what I consider to be its most valuable attribute. Redirects are commands instructing shell programs to output into a separate file (either extant or created at the time of the redirect); redirects can also work in reverse, e.g. reading a file to a program. Pipes are slightly different and are intended to transform outputs of a command or a program into inputs for other commands or programs. Daisy chaining redirects and pipes lends itself well to modular programming and shell scripting as we'll explore in further sections like 5.

Variables speak for themselves, they're the backbone of most shell scripts and can be stored and called easily in the shell for numerous uses. While and For loops speak for themselves as well and allow incremental operations.

In the following sections, we will explore a quick example of using a shell script to create an SBATCH file for use in supercomputer program execution, the script uses variables to store values and writes them into a file using a redirect and the cat command. We will also explore various easy implementations of while and for loops. In the next chapter we will see how to put all of these tools, as well as tools we've learned up to this point into scripts that perform useful operations like web scraping, multiple option selection and execution, and data handling and normalization.

4.1 Redirects and variables

Redirects are somewhat self explanatory in that they allow they redirection of one thing into another. There are two types of redirects, the redirect of input < and the redirect of output >. A redirect of input can be imagined as the following: program $\leftarrow$ file. A redirect of the output can be imagined as the following: program $\rightarrow$ file

In the redirect of an input, a file's contents can be loaded into a program for analysis or other handling with that program. In the redirection of an output, a file can have its contents written to with the output of another program.

A use case of the redirect of input < is given in the following where we determine the number of lines in a large text file using the wc (word count) command with the -l (lines) option and we are redirecting the large text file (in this case the Webster's Dictionary) into wc with the redirect of input. The result will be the number of lines in the Webster's Unabridged Dictionary being printed to the terminal:

```
uname@uname:~> wc -l < Websters_Unabridged_Dictionary.txt
  974267
```

A very popular use case (at the very least in this text) of the redirect of output is in writing files based on what is returned by a shell program. This a very valuable tool when you are aiming to create new files based on the contents of a script and the variables inside of it. Please see the following terminal session where we have a number of text files that and we redirect the output of the ls command into a new file called Text_Files.txt which will contain the names of all the original files in the directory:

```
uname@uname:~> ls
Text_File_1     Text_File_2     Text_File_4     Text_File_6     Text_File_8
Text_File_10    Text_File_3     Text_File_5     Text_File_7     Text_File_9
```

```
uname@uname:~> ls > Text_Files.txt
uname@uname:~> head Text_Files.txt
Text_File_1
Text_File_10
Text_File_2
Text_File_3
Text_File_4
Text_File_5
Text_File_6
Text_File_7
Text_File_8
Text_File_9
uname@uname:~> ls
Text_File_1   Text_File_2  Text_File_4  Text_File_6  Text_File_8  Text_Files.txt
Text_File_10  Text_File_3  Text_File_5  Text_File_7  Text_File_9
```

The general idea of variables is pretty self explanatory. Variables can be quickly implemented in BASH, just like many other programs for all sorts of uses which include but are not limited to storing mathematical values, storing the number of an iteration in a loop, storing a string of text, storing the location of a file or executable, ect ... Variables in the shell are first defined with a string followed by an = sign and then the variable can be called later on by referring to the string with a $ before the string. Please see the following terminal session for how to create and call variables in some instances (for more coverage of the program bc please refer to section 3.12, and for more coverage of the pipe special character, please refer to section 4.2):

```
uname@uname:~> a=1
uname@uname:~> echo $a
1
uname@uname:~> b="Hello World"
uname@uname:~> echo $b
Hello World
uname@uname:~> lat=1.000000
uname@uname:~> Top_Atom="5.40425600"
uname@uname:~> Titanium_Nitride_Dimer_Bond_Length="1.6337"
uname@uname:~>
uname@uname:~> lat_a=$(echo "scale=9; $Top_Atom + $lat" | bc)
uname@uname:~> lat_b=$(echo "scale=9; $lat_a + $Titanium_Nitride_Dimer_Bond_Length"
    | bc)
uname@uname:~> echo $lat_a
6.40425600
uname@uname:~> echo $lat_b
8.03795600
```

Redirects and variables are essential ingredients in most of my shell scripts. The following is a fragment from a script that I frequently use for submitting calculations to a supercomputer which uses redirects and variables. The script is divided into two sections. The first section defines variables using the following formalism: varname='var-value'. All of these variables can be called at any time in the shell after they are stored. Even a the location of an executable can be stored and then used. Here we direct BASH to the executable for VASP but the same script can be used interchangeably with any executable really. This is immediately useful because now you do not need to handle rewriting and creating individual files any more but you can compose numerous files into a single script and have that script create your files for you in the shell before executing the code (especially when that code is dependent on the files that were written with the cat command in the shell) that you are interested in.

As just discussed, redirects < and > (also « and ») are known as the standard input and standard output commands respectively. Because redirects let you redirect the output of one program directly into another, the shell has a powerful ability to compose the use of many other programs in a step-wise manner. Like the > command which can be used to direct the output of command into overwriting a file, the » command can be used similarly but instead of overwriting the file, the output will be appended to the file (e.g., written to a new line or set of lines in the file). There is a useful distinction to know however, using > will take an entire standard output and write it to a file. Using the command 2> will instead just allow the user to redirect the exit code of the file or the output of the command into the file. Each of these commands have many uses and are good tricks to have.

The second section of this script is slightly more complicated and uses a pair of redirects. Of these, the first line (line 17 in the overall script) is the most complicated and does the heavy lifting. We first use cat (the familiar concatenate command) and redirect its standard output into a new file that is defined partially as a variable. The file is written explicitly as the variable Job_Name and then appended with the .sb extension which is used for sbatch commands (a tool used frequently for queuing programs to be run on a supercomputer). Next, cat redirects the standard input of the terminal session into the file now being called Sbatch_Example.sb (since it will call the value of the variable Job_Name) until it comes to the characters EOF whereupon it will terminate cat and write the file.

After the line containing the cat command and the redirects, we can see that the following lines contain the shebang line for a BASH script, several SBATCH commands, and then it issues the srun command for specifying certain parameters to be used in a calculation. EOF is written to signal cat to halt writing the file and then the script uses 'echo' to write an update to the terminal stating that the file has been created.

```
1  #!/bin/bash
2  #-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
3  #-:-:-:-:- Create variables for storing SBATCH commands -:-:-:-:-:-
4  #-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
5  Job_Name="Sbatch_Example" # Give a name to apply to files
6
7  Job_Time="00:30:00"   # Give the run time in hh:mm:ss
8  Job_Nodes="64"        # Give the number of calculation nodes
9  Job_Queue="debug"     # Give the queue (e.g., 'debug' or 'regular')
10
11 Module_Location="~/uname/codes/vasp/vasp.5.3/vasp"
12
13 #-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
14 #-:-:-:-:-:-:-:-:-: Create an SBATCH Script :-:-:-:-:-:-:-:-:-:
15 #-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
16
17 cat > ${Job_Name}.sb << EOF
18 #!/bin/bash
19 #SBATCH --job-name=$Job_Name
20 #SBATCH -N ${Job_Nodes}
21 #SBATCH -C haswell
22 #SBATCH -q ${Job_Queue}
23 #SBATCH -t ${Job_Time}
24
25 srun -n32 -c2 --cpu_bind=cores ${Module_Location}
26
27 EOF
28
```

```
echo " Writing input file ${Job_Name}.sb..."
echo "  done"
```

This methodology becomes very helpful when you run many scripts of the same type with small variations between them, especially when they are being run on remote machines like a cluster. Having your scripts do the tedious work for you is mainly the whole point of shell scripting (at least for most of my use cases). Redirects and variables take care of much of this.

4.2 Pipes: an example with the tee command

Pipes are another staple of shell scripting. All that pipes do is allow you to use the output of one program as the input of another program. The implications of how to use pipes then is vast. We won't go into extreme depth with pipes here but I'll give what I find to be a very useful example.

Suppose you have a script that prints some useful output to the terminal. An example of this would be, for example, the status of a program that you are running on a supercomputer cluster. Sure it is useful to see that output with your eyes, for example to see how long it has taken a calculation to be completed in real time but that information may be useful in the future as well. Saving the output of a terminal session then becomes very interesting and important for future reference. One way this can be done is with a pipe and the program called 'tee'.

The following BASH command executes a shell script called File_Name.sh (again using the ./ command) and then uses a pipe | to direct the terminal output into a program called tee (here we also use & after the pipe, this allows us to pipe the standard output as well as the standard error of the program into tee). tee is interesting but briefly its use here is to not interfere with what is printed to the terminal while simultaneously writing everything printed to the terminal to a file. The -a option which we give to tee tells tee to append to a file (in this case we give it the file README.txt). So, as long as the program File_Name.sh is running, tee will be writing the output of the terminal to the README.txt file. This can be halted by issuing ctrl+c to the terminal.

```
./File_Name.sh |& tee -a README.txt
```

4.3 Loops and built-in logical counters

Loops are the bread and butter of shell scripting for scientific computing because, very often, you are intending to determine some quality of a physical system with respect to another quality of a physical system. For example, maybe you are trying to determine the temperature as a function of time as given by a weather station. You can use a loop to continuously collect information from the weather station as a function of time (for an in-depth coverage of this as a web-scraping project, please see section 5.3). Or perhaps you might want to use the shell to control the value of a lattice parameter in the input files of a DFT code and run calculations for many different lattice parameters in a specified range (for more coverage of this topic, please see chapter 9. There are too many instances of useful applications of loops to list here.

In the following sections, we will cover for loops, while loops, until loops, and if statements. In section 5.1 we will cover more on the use of the BASH equivalent of switch/case statements, and in 5.4 we will cover more on the use of functions in BASH.

4.3.1 For loops

For loops are another familiar element in most programming languages. As with others, for loops in BASH allow the step-wise execution of code based on a sequence that controls the loop by assigning the value of that sequence at a specific step to a control variable for each step in the loop, continuing until the conditions of the loop are exhausted.

An introductory demonstration of for loops can be made by giving a small sequence which counts from 0 to 5 in increments of 1 and creates a directory named after the current step in the loop for each increment along that sequence. Please see the following example:

```
uname@uname:~> for a in {0..5..1}; do mkdir $a; done
uname@uname:~> ls
0   1   2   3   4   5
```

We can also structure for loops recursively

```
uname@uname:~> for a in {0..5..1} ; do mkdir $a; cd "$a" ; for b in {0..5..1} ; do
    mkdir $b; cd "$b"; cd ..; done; cd ..; done
uname@uname:~> ls -r *
for_loop.sh

5:
5   4   3   2   1   0

4:
5   4   3   2   1   0

3:
5   4   3   2   1   0

2:
5   4   3   2   1   0

1:
5   4   3   2   1   0

0:
5   4   3   2   1   0
```

Iterating on the immediately previous example, we can create our own recursive listing program with the use of a for loop! Please see the following script that recursively lists the contents of all the directories within a directory, this is roughly equivalent to the 'ls -r * command that we just used:

```
uname@uname:~> ls
0   1   2   3   4   5
uname@uname:~> for a in */; do echo "$a:"; cd "$a"; ls; cd ..; done
0/:
0   1   2   3   4   5
1/:
0   1   2   3   4   5
2/:
0   1   2   3   4   5
3/:
0   1   2   3   4   5
4/:
0   1   2   3   4   5
```

```
5/:
0   1   2   3   4   5
```

The for logical operation is especially valuable in scientific computing with shell script because it can allow for automation in the creation of calculation scripts. That is to say, for many programs like VASP, Quantum ESPRESSO, Exciting, etc... input files can be turned into templates and tweaked slightly and iteratively to give a whole series of calculations that describe many atomic configurations of a system. This is especially evident in VASP calculations where the POSCAR file has a universal scaling constant for the lattice vectors that can be turned into a shell script controlled variable.

Lattice vectors and their scaling directly impact many properties of materials: a direct result of the distances between ions and their configuration(s) in a crystal being the progenitor of many properties. Lattice vectors and mostly anything else you can imagine can be controlled using variables in shell scripts so that properties as a function of other properties can be calculated (swept) over a space. One pertinent example of this is the calculation of total energy in a crystal over a range of structures and lattice vectors to determine the global minimum energy atomic configuration which usually corresponds to a stable or preferred structure.

An implementation of exactly this methodology will be discussed in chapter 9 but a slight teaser of how the for loop is constructed is as follows:

```
for lat in `seq -w ${Lat_Param_Min} ${Step_Size} ${Lat_Param_Max}`; do
```

Remember that, in the previous example, backticks ` are a very important part of the operation as the backtick ` tells BASH to execute all code within backticks ` before running a program on the code contained within the backticks. Choosing between backticks and parentheses for a certain application can sometimes be ambiguous as they can have similar results. However, that is a bit beyond the scope of the present section.

For loops can also be paired with the modulo operation % which returns the remainder of an integer division operation. In the following example (which was inspired by an article on www.shell-tips.com) we can check for even and odd numbers in a sequence

```
uname@uname:~> for a in {1..10}; do if (( a % 2 )); then echo "\$a=$a: odd"; else
    echo "\$a=$a: even"; fi; done
$a=1: odd
$a=2: even
$a=3: odd
$a=4: even
$a=5: odd
$a=6: even
$a=7: odd
$a=8: even
$a=9: odd
$a=10: even
```

4.3.2 Until loops

As the name might imply, until loops allow for a process or series of processes or commands to continue some condition is satisfied. This can in some regards result in similar effects to for loops and while loops but has its own distinct advantages in some aspects. As a primer, please see the following command which uses an until loop to create distinct, empty text files and then fills them with some distinct text. The loop can be modified to create any n number of files controlled by a variable \$a which is incremented based on the logical counter operation $((a = a + 1))$:

```
uname@uname:~> a=1; until [ $a -gt 5 ]; do touch $a.txt; echo "Hello! My name is $a.
    txt!" > $a.txt; ((a=a+1)); done
uname@uname:~> ls; cat *
1.txt   2.txt   3.txt   4.txt   5.txt
Hello! My name is 1.txt!
Hello! My name is 2.txt!
Hello! My name is 3.txt!
Hello! My name is 4.txt!
Hello! My name is 5.txt!

```

As just stated, the operation $((a = a + 1))$ is known as a counter and does the heavy lifting in the previous commands. In BASH as well as many other languages, $+$ is known as the increment operation and $-$ is known as the decrement operation; both of these are built-ins in BASH and they comprise just two of a larger set of logical operations which are broadly useful.

As you may imagine, until loops can have interesting consequences if they are paired with a logical comparison like true and false. For example, if we were to say do x while y is true then there may be a case when y is always true. This is a neat way of making an infinite loop in BASH. Please see the following example implementing an infinite loop with an until statement in BASH where the command will echo the phrase ENDLESS to the terminal until the condition 'false' is satisfied (which it never will be in this case):

```
uname@uname:~> until false; do echo ENDLESS; sleep 1; done
ENDLESS
ENDLESS
ENDLESS
ENDLESS
^C
```

4.3.3 While loops

While loops are common in many languages and just allow the interpreter to run a series of commands while something else is happening. I find this particularly useful when creating infinite loops to perform some operation or having a loop terminate after some condition has been met. One of the most streamlined ways of creating an infinite loop with a while loop is the following example where the option we give to the while loop is : which, in this context, is the special character for the null operator which essentially says 'don't do anything at all':

```
uname@uname:~> while : ; do echo ENDLESS; sleep 1; done
ENDLESS
ENDLESS
ENDLESS
^C
```

While loops can be paired with a logical operation as their option and thereby be used to create a loop that will commence until some algebraic expression has been satisfied. Simple algebraic expressions like greater than $>$, and less than $<$ can rapidly be given to while loops as their option too. The following example pairs a decrement operation $-$ with the greater than operation saying 'while this variable is greater than zero, continue and for each step subtract 1 from the variable until the conditions are exhausted:

```
uname@uname:~> x=5; while (( x > 0 )); do echo $((x--)); done
5
```

```
4
3
2
1
```

We can perform a similar operation, this time using the increment operation ++ instead and pairing it with the less than operation:

```
uname@uname:~> x=-5; while (( x < 0 )); do echo $((x++)); done
-5
-4
-3
-2
-1
```

We can perform this same sort of logical arithmetic in more complicated cases than simple addition or subtraction of 1 and instead operate with an arbitrary argument of the increment/decrement value. The argument of the increment and decrement value is given in the second to last command in the following examples with let "var-=n" where n is an arbitrary quantity and the -= operation can be substituted for whichever valid operation suits your use case.

In the following example, we give a variable with initial value 100 and then decrement that value by 25 for each step of the while loop. The conditions of the while loop say: while the variable is greater than or equal to (the -ge option) 25, continue.

```
uname@uname:~> var=100; while [ $var -ge 25 ]; do echo Number: $var; let "var-=25";
    done
Number: 100
Number: 75
Number: 50
Number: 25
```

We can also run the exact same loop as above swapping out the semicolon after the do echo command with a double ampersand && which says 'do the next step only if the previous step has completed successfully':

```
uname@uname:~> var=100; while [ $var -ge 25 ]; do echo Number: $var && let "var-=25"
    ; done
Number: 100
Number: 75
Number: 50
Number: 25
```

Keeping along a similar theme, we can change the argument of the greater than or equal to parameter my modifying the numerical quantity within the square brackets like in the following. This time, we say 'while a variable with initial condition of 100 is greater than or equal to -100, decrement that variable by 25 for each step of the loop:

```
uname@uname:~> var=100; while [ $var -ge -100 ]; do echo Number: $var; let "var-=25"
    ; done
Number: 100
Number: 75
Number: 50
Number: 25
Number: 0
Number: -25
Number: -50
Number: -75
Number: -100
```

We can swap the type of operation being performed within the square brackets quickly to less than or equal to by changing the operation to -le. Then we can run a similar loop as before, this time saying 'given a variable with initial value -100, while that variable is less than or equal to 25, increment that value 25 for each step of the loop:

```
uname@uname:~> var=-100; while [ $var -le 25 ]; do echo Number: $var ; let "var+=25"
    ; done
Number: -100
Number: -75
Number: -50
Number: -25
Number: 0
Number: 25
```

One useful case of this is the following example. We can use while to continuously print as output to the shell a the status of a program, directory, file, etc... See the following command:

```
while [ 1 ] ; do clear; ls -lrt; sleep 5; clear; done
```

In this example we use while [1] ; do as the beginning of this command. While [1] will loop all of the commands following the semicolon and after issuing the do command, right up to when the done command is issued. Parameters within the square brackets can be modified to whatever the user really desires; however, here the [1] just makes for a convenient way of creating an infinite while loop. Following the issue of while [1] ; do ... we have ls -lrt; sleep 5; done. The idea is that we are just continually updating the terminal with the contents of the current directory, waiting for 5 seconds, clearing the terminal, and then doing it all again.

This can be done in a more immediately useful way (especially when paired with a pipe to the tee command as explained in section 4.2) by issuing the sqs command (which is a specific squeue command for some clusters) that prints the status of a program running on a queue in a super computer. We can use the while command in the following way then to collect information about what the status of a program running on a cluster is:

```
while [ 1 ] ; do sqs; date; sleep 5; done
```

This program also calls the date command which prints the current system time to the terminal. The output of this small program may look like the following which is a convenient way to monitor a program in real time (some of the output of sqs has been truncated so that the output fits more appropriately in this printed text):

```
JOBID     ST USER  NAME NODES TIME_LIMIT TIME  SUBMIT_TIME
45727798  R  uname TiN  16    10:00      0:02  2021-08-16T18:18:43
Mon Aug 16 18:18:48 PDT 2021
JOBID     ST USER  NAME NODES TIME_LIMIT TIME  SUBMIT_TIME
45727798  R  uname TiN  16    10:00      0:07  2021-08-16T18:18:43
Mon Aug 16 18:18:53 PDT 2021
JOBID     ST USER  NAME NODES TIME_LIMIT TIME  SUBMIT_TIME
45727798  R  uname TiN  16    10:00      0:12  2021-08-16T18:18:43
Mon Aug 16 18:18:58 PDT 2021
```

4.3.4 If statements: check whether a program is installed

If is another operator to which anyone with programming experience will most likely have had some exposure. As a primer to shell scripts, we can consider the following (a useful little code block that checks for whether a program is installed on your system which was adapted from some initialization scripts from the library of Quantum Espresso example calculations):

```
# check for gnuplot
gnuplot_command=`which gnuplot 2>/dev/null`
if [ "$gnuplot_command" = "" ]; then
        $ECHO
        $ECHO "gnuplot not in PATH"
        echo "Installing Gnuplot"
  sudo apt-get install gnuplot
  else
  echo "Gnuplot installed"
fi
```

You may peruse the program at your leisure as we will talk in more depth about this exact block of code in the next section, however it uses the conditionality of the if statement to out benefit. The script sets a variable (presented in backticks which tell BASH to evaluate everything within the backticks before evaluating the remainder of the line of code, so make sure that your system is rendering ` as a backtick and not an apostrophe) that outputs the 'which' status of the program called Gnuplot. Then we construct an if-statement which checks whether there was something written to the 'gnuplot_command' variable (it compares the contents of the variable to the empty contents ''). In the case that there was something written, then the program will echo to the terminal that Gnuplot is installed; in the alternative case (using the else part of the if-statement) the program finds the variable 'gnuplot_command' to be empty, whereupon it will print some text to the terminal and install the program.

You can run this program for yourself by following these steps:

1) Create a file called Installation_Check.sh

```
touch Installation_Check.sh
```

2) Make the file called Installation_Check.sh executable

```
chmod +x Installation_Check.sh
```

3) Begin editing the program called Installation_Check.sh with vim

```
vim Installation_Check.sh
```

4) Paste the above program into the Installation_Check.sh file using ctrl+v and then pressing the escape button followed by :wq which will save your changes and exit vim

5) Run the program to check whether Gnuplot is installed:

```
    uname@debian-micro:~$ ./Installation_Check.sh
Gnuplot installed
```

As you can see, we do indeed have the Gnuplot program installed as the terminal has returned the 'Gnuplot installed' line from our script.

In the next section, we will explore several scripts that I find to be useful and which use most or all of what we have covered up to this point to perform useful operations and automation.

5 Shell Scripting: Making General Purpose Tools

Shell scripting has broad utility in a range of situations involving computational tasks that lend themselves well to automation. Mainly, as we've seen previously in this text, the shell is used to reduce the total effort required to perform what might be otherwise excessively tedious operations if performed with a graphical user interface (GUI), mouse, and keyboard. We've already seen how useful the shell is in performing such tedious operations in the section concerning wget and curl (3.6). In several computational physical sciences, a very common implementation of shell scripting is calculating properties of materials with molecular dynamics (MD), density functional theory (DFT), (I shutter to say it but due to the computational cost) GW, and all sorts of quantum chemistry, theoretical spectroscopy, and others I'm undoubtedly forgetting.

In the next few subsections, we'll explore some shell scripts as easy to follow 'recipes' that I think are worthwhile to consider for a variety of reasons. All efforts are made to explain the functionality of these scripts point by point and make them tunable to the reader's needs.

5.1 Make a curses-style text-based user interface with while loops and case functions

User interfaces (UIs) are clearly ubiquitous but design of a graphical user interface (GUI) can be tedious and unnecessary for many applications. Especially in 'headless' systems, GUIs just would slow everything down for many operations. Consider the older-style BIOS text-based user interface that many of us are accustomed to, it can be navigated quickly and clearly understood without having nearly any footprint on system resources. Curses, and ncurses are text-based user interface (TUI) libraries that are very useful for programs which need some user instructions but where a GUI might be overkill. There's clearly also some strange sort of style points (at least in this author's opinion) to be awarded for the use of ncurses.

Let's consider how to make a text-based user interface with the shell using a function, some variables, and a whole bunch of echo statements. This menu program was partially inspired by an underappreciated YouTube BASH tutorial called "Creating Command Line Menus with Shell Scripts" by theurbanpenguin. Please see the following script:

```bash
#!/bin/bash

# menu.sh
# Written by Steven Bopp on 18 May 2016

function selection () {
echo -e "\n"                    # Add a new line with \n
echo -e "Enter your selection \c" # Suppress a new line with \c
}

while true
do
    clear # Erase the previous input each time this block is run
        # Display menu text
        echo "=========================================="
        echo "===== Programs and Tools Launch Menu ====="
        echo "=========================================="
        echo "Enter 1 to launch programs"
        echo "Enter 2 to access system tools"
        echo "Enter 3 to manage Linux packages"
        echo "Enter q to exit this menu"
        selection

    read answer_zero
    case $answer_zero in # Start primary switch case block

1) # This block launches the program launch menu
        clear
        echo "==========================================="
        echo "========== Program Launch Menu ==========="
        echo "==========================================="
        echo "Enter 1 to launch VESTA"
        echo "Enter 2 to run nano on an existing file"
        echo "Enter 3 to run the Ranger file explorer"
        echo "Enter q to exit the menu"
        selection

    read answer_one
    case $answer_one in
        1) # This block executes VESTA in Ubuntu.
            cd /home/uname/Documents/Programs/VESTA/VESTA-x86_64
            ./VESTA ;;
        2) # This block will execute nano on a user input file name
            echo -e "Enter the name of the file \c"
            read textfile # Read user input and store as $textfile
            echo "You are now editing $textfile"
            nano $textfile ;; # Execute nano on $textfile
    3) # Execute Ranger
        ranger ;;

    esac # End program launch sub-menu switch case block
    read input_one
            ;; # End program launcher

2) # This block launches the system tools sub-menu
        clear
        echo "==========================================="
        echo "======= System Tools Launch Menu ========="
        echo "==========================================="
        echo "Enter 1 to launch htop"
        echo "Enter 3 to run Nmap on a specific IPv4 address"
        echo "Enter 4 to run Address Resolution Protocol"
        echo "Enter 8 to restart the network-manager service"
        echo "Enter q to exit this menu"
        selection

    read answer_two
    case $answer_two in
        1) # This block launches htop
            htop ;;
        3) # Execute Nmap on a user input IPv4 address
            echo -e "Enter the IPv4 address \c"
            read nmap
            echo "Nmap will now scan the given IPv4 address: $nmap"
            Nmap $nmap ;;
        4) # Run local network IPv4 Address Resolution Protocol
            echo "Host address:"
            hostname -I
            echo "Network addresses:"
            arp -a ;;
        8) # This block will restart the network-manager service
            sudo service network-manager restart ;;
        q) # This block executes the exit command from this menu
            exit ;;

    esac # End of system tools launch sub-menu
    read input_two
            ;; # End system tools sub-menu

3) # This block manages packages on Linux
        clear
        echo "=========================================="
        echo "======== Manage Linux Packages ============"
        echo "=========================================="
        echo "Enter 1 to see install policy of a program"
        echo "Enter 2 to search for installed printer packages"
        echo "Enter 3 to delete configuration and/or data files + dependencies of a package in Ubuntu"
        echo "Enter q to exit this menu"
        selection

    read answer_three
    case $answer_three in
    1) # Run sudo apt-cache policy on a program
        echo -e "Enter the name of the program \c"
        read program    # Read user input and store as $program
        echo "You are now editing $program"
        apt-cache policy $program ;; # run on $program

    2) aptitude search printer | grep ^i ;;
    3) echo -e "Enter the name of the program \c"
        read programa   # Read user input and store as $programa
        echo "You are now purging $programa"
            sudo apt-get purge --auto-remove $programa ;;
    q) exit ;;

    esac
    read input_three
     ;; # End of package manager


    esac # End primary switch case block
    echo -e "Enter return to continue \c"
    read input_zero # New variable called input from the case statement which is displayed

done
```

This script gives us the unique capability of creating a text-based user interface with the shell. We can use this for all sorts of things, most frequently I use it for automating programs in the shell and remembering long or tedious commands. I especially find this useful when I run a program infrequently and would forget the specific command(s) to run said program if they were not written down somewhere, wasting time. So, to combat that, I just usually add entries to a large, master menu program where I store many commands and series of commands that I would have a hard time remembering otherwise but I know I'll come back to at some point. We can run the above menu script (after making it executable with chmod; for more on ownership and executability, see section 2.4) as follows:

```
MainUsers-iMac:Desktop mainuser$ chmod +x menu.sh
MainUsers-iMac:Desktop mainuser$ ./menu.sh
```

The program will output a series of 'screens' which can be navigated by using the number keys. Running the above command will give the following and wait for the user's input:

```
==================================================
====== Programs and Tools Launch Menu ======
==================================================
Enter 1 to launch programs
Enter 2 to access system tools
Enter 3 to manage Linux packages
Enter q to exit this menu


Enter your selection
```

Entering option 3 (for example) will result in the display of the following screen:

```
==================================================
========== Manage Linux Packages ==========
==================================================
Enter 1 to see install policy of a program
Enter 2 to search for installed printer packages
Enter 3 to delete configuration and/or data files + dependencies of a package in
    Ubuntu
Enter q to exit this menu


Enter your selection
```

As you can see, this is a remarkably simple way to collate all sorts of programs and commands into one agile and easy to modify script. The individual blocks can be modified to have any content you wish and the 'levels' of navigation can be increased with the addition of more blocks within other blocks. Modularity is 'baked in' here intentionally to make it simple to change functionality rapidly and without having to remember complicated scripting commands.

The only shortcoming of this script is that its all controlled with a single while loop. This means that if we wanted to do something like deploy the script with additional options (like what you'd see in other shell programs e.g., -v, -i, etc...) we couldn't do that without significant hassle. Later on in this text, I'll show how to create a similar menu which has the ability to be run with options based on the program called getopts and construction of the code using functions instead of while loops. Implementations of scripts with getopts will be discussed at length in section 5.5.

5.2 Make a timer in the shell (for use within scripts) using variables and built-in math functions

One useful trick in the shell is to create a timer inline with the BASH commands. This is helpful especially when you are running programs locally and want to see how long each successive step has taken. I use this methodology when running Quantum Espresso calculations on my local machines because I like to get a baseline reading of how long calculations will take to converge before passing them to a remote supercomputer.

The timer script is simple and supplied with a lot of placeholders where it is intended that you can paste your own code. All this script does is to create a master variable called 'START_TIME' which stores a number of seconds. After this, at each step in the run of the script, we use the simple mathematical functionality in BASH built-ins to calculate the elapsed seconds between the start time and the current time, store that output as a new variable, and then echo the value of that variable to the terminal. Please see the following script:

```bash
#!/bin/bash

START_TIME=$SECONDS # Begin elapsed time measurement
# Timer here inspired by Tom Anderson from StackOverflow

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-: Begin Calculation Directions -:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

# Insert some shell code here, perhaps a directions for the script

ELAPSED_TIME1=$(($SECONDS - $START_TIME))
echo "  It has been $ELAPSED_TIME1 seconds"

echo "    Task 1 complete"; sleep 1

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-: Begin Non-Self-Consistent Calculation :-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

# Insert some shell code here, perhaps a calculation

ELAPSED_TIME2=$(($SECONDS - $START_TIME))
echo "  It has been $ELAPSED_TIME2 seconds"

echo "    Task 2 complete"; sleep 1

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:- Begin epsilon.x Calculation -:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

# Insert some shell code here, perhaps a calculation

ELAPSED_TIME3=$(($SECONDS - $START_TIME))
echo "  It has been $ELAPSED_TIME3 seconds"

echo "    Task 3 complete"; sleep 1

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:- Create Au_Permittivity.dat :-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

# Insert some shell code here, perhaps creating a file with cat

ELAPSED_TIME4=$(($SECONDS - $START_TIME))
echo "  It has been $ELAPSED_TIME4 seconds"

echo "    Task 4 complete"; sleep 1

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
```

```
#-: Begin Plotting with Gnuplot (Assuming That It is Installed) :-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

# Insert some shell code here, perhaps plotting something...

ELAPSED_TIME5=$(($SECONDS - $START_TIME))
echo "  It has been $ELAPSED_TIME5 seconds"

echo "    Task 5 complete"; sleep 1

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:- End Elapsed Time Measurement :-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

ELAPSED_TIME=$(($SECONDS - $START_TIME))
echo "  It has been $ELAPSED_TIME seconds"

echo "    Job Completed!"
```

In this example script, we use the BASH built-in called sleep. All sleep does here is to delay the next operation in the script for a set amount of time. Sleep can be a very useful command in loops which do not need to have a continuous duty cycle like what we will see when we discuss web scraping with the shell in section 5.3. Running this script after making it executable with chmod as discussed in section 2.4 (with the sleep function as a dummy placeholder for some other operation) returns the following output to the terminal:

```
MainUsers-iMac:Desktop mainuser$ chmod +x timer.sh
MainUsers-iMac:Desktop mainuser$ ./timer.sh
  It has been 0 seconds
    Task 1 complete
  It has been 1 seconds
    Task 2 complete
  It has been 2 seconds
    Task 3 complete
  It has been 3 seconds
    Task 4 complete
  It has been 4 seconds
    Task 5 complete
  It has been 5 seconds
    Job Completed!
MainUsers-iMac:Desktop mainuser$
```

5.3 BASH web scraping HTML using Gnuplot, lynx, awk, sed, and bc (and checking for installed programs with if statements)

Web scraping has become a super popular topic lately for all sorts of applications involving the automated collection of data. Stock trading and price monitoring are one very interesting example of this technology but, as an easy way to introduce web scraping, we will consider the collection of weather information from a micro weather station. In order to do our web scraping here, we will rely on a few shell programs including lynx, awk, sed, and bc (the bc calculator is discussed at length in section 3.12). Furthermore, we will visualize the results completely headless in the terminal with the Gnuplot software package (discussed more in section 3.8).

Absolutely none of the following text (or any of the text within this document) is intended to be legal advice!

Do your own research and make sure that your implementation(s) of web scraping are 100% legal. Web scraping is, in general terms, the process of automatically collecting information from the internet (or a network) in general and/or an individual website for the purposes of cataloging, databasing, owning, re-distributing, or using in any means information relevant to the interests of the collector. Web scraping may be a bit of a gray area at the current time, legally speaking. It *appears* to be legal to use instrumented browsers or scripts to collect data from any legally and publicly accessible website so long as the contents are not malicious or illegal and do not interfere with copyright or other rights held by the owners of the content; *however, I am not a lawyer and cannot verify or deny these the legality of things.*

The reason I bring this up is that there seems to be a blurred line between distributed denial of service (DDoS) attacks and web scraping in the case when web scraping is done at very high refresh rates. That is to say, if your web scraper is asking a server for lots of information many times per second or per minute and it is inadvertently (or intentionally) hogging lots of that server's resources to the detriment of that server or its owner, then that script is having a similar effect as a DDoS attack. At the current time, as far as I can tell, DDoS is an illegal operation. My recommendation is to not do illegal things, and especially to not use the information contained within this text to do illegal things.

The web scraping script supplied in the following is divided into four sections that all have their own specific uses. The first section is easiest to describe. All we do here in the first section is to check that lynx, bc, and Gnuplot are installed and, in the event that they are not, install them. The way this is achieved is through three if/else statements that individually use the BASH built-in called 'which' to check whether a program has been installed in the path. The value of where that program is (or is not) within the path is then stored as a variable where it is compared conditionally to an if/else statement. Nothing much happens in the if/else statement if the program is installed but in the event that the program is not installed, the 'else' portion of the statement will install the program.

After that (still in the first section of the script), I use the built-in called timedatectl to change the time zone to be PDT (which is the timezone in which the weather station that we'll be scraping is located). Finally in this section, I create a variable to store the current data and then pass that variable to a new variable called File_Name which will always be unique for each new run of the script because it will always be prepended with the unique date on which the script was run. There is also a section of code near the very beginning that is commented out, I use this sort of header as a copy and paste on most of my scripts when I am working on a remote machine. The purpose of this chunk of code (please leave it commented out) is to create a new file that can contain the script which follows. This is very useful on remote machines because you can copy and paste the line of code (without the comment mark) into the terminal on the remote machine and then paste your script into vim, write and quit from vim, and then the script will be run automatically. Note that the script will be run with no hangups (nohup) and that there is a fork at the end of that commented line of code so the program can run in the background.

In the second part of the web scraper script, all we are doing is creating a new empty file which will be named based on the value of the variable ${File_Name} which we defined in the previous section of code. This is done quickly using cat and redirecting all of that output until cat sees the string EOF (for more on cat and similar commands, see section 2.2.2). Additionally, for archival purposes, the data file has a header added which tells the contents of each of the columns and their units which are going to be written into the file. It is important that the header of this file is contained within quotation marks because that way it will not be considered by plotting programs like Gnuplot. Finally, just as an aesthetic touch (and to see that the program has reached a certain 'checkpoint'), I have the script echo to the terminal that it is writing the file called ${File_Name}.

In the third section of the web scraper script, we create a program that can plot the data contained within the data file we previously created. The name of this script is Weather_Plotter.sh and will be able to be run with ./Weather_Plotter.sh after we run the main web scraper script. If you are familiar with Gnuplot, then this may be relatively straightforward to you, there are however some aspects that I customize to make the functionality of the plotting program more functional. We use cat the normal way to redirect the following output, except in this instance we put EOF into quotation marks so that BASH will not mess with any of the variables contained inside the script. Then we create two variables that are made by searching the data file we are scraping into using the awk command (somewhat humorously this could be called a scraper scraper program maybe?). Awk here is used to check the data file for the UNIX times corresponding to the start time of the data and the end time of the data, these are useful for making the data plotting more streamlined. What follows until the EOF string is a script that Gnuplot will recognize. This script that Gnuplot will recognize is created again using cat, however we use the exclamation mark as the end condition so as to not interfere with the "EOF" end condition of the parent script.

The Gnuplot script is relatively straightforward but contains a few nice tricks that I will explain here. Since I run these scripts on a remote machine in the Google cloud without any GUI, and because I want to be able to visualize the data quickly, we want to be able to make plots and visualize them all within the terminal and not have to use a GUI program. Gnuplot has a terminal style called 'dumb' which allows us to do exactly that. The 'dumb' Gnuplot terminal will give us a direct output to the terminal of the plot we ask for based on the data that we give to Gnuplot. We have to tell Gnuplot that the data going into the x-axis is going to be time and then we set the format to seconds using the command "%s". I do this specifically because I think it's easy to work with UNIX time which is all in seconds since January 1st, 1970 at UTC. The UTC part of the UNIX time is annoying to me however so later on I covert this in the code to Pacific time so that the plot won't have a rigid offset relative to my timezone. I then set the x-axis of the plot to have a specific data labeling where the month and the day appear just above the hour and the minute. The fanciest part of this script is that we then use the previously defined start and end date variables that we collected with awk to automatically set the x-range in the script. After that we plot the data from the file using the 'p' function in Gnuplot (which could be replaced by the 'plot' function, 'p' is just the shorthand which is nice most of the time) using lines and we also suppress the title of the data set in the final plot. Finally we write that to the disk and change the script's mode so that it is executable.

Finally, the fourth section of this script titled as 'Scraping Loop' is where the all of the scraping is actually done. Everything up to this point has just been setup. To start, this part of the script prints some status text to the terminal and then begins a while loop that will run infinitely or until the program is killed. This while loop contains all of the remainder of the code and is set to run and then re-run with an interval of 10 minutes which is defined by the sleep function at the end of the script. The while loop starts by using lynx to dump the contents of an HTML page into a nicely formatted .txt file. An output of this lynx command can be seen below, please note that there is a lot of extra text at the end of the file that we don't specifically want (everything after 'References').

```
debian-micro@debian-micro:~$ cat lynx_dump.txt
   REFRESH(61 sec): [1]https://hpwren.ucsd.edu/Sensors/SDSC/
   HPWREN multicast-based weather station data display
    20210822 19:47:40 - UCSD San Diego Supercomputer Center: 32.88N 117.24W
   400'
   This sensor is a Davis met station
   graphs are since midnight two days ago
   [2]Outside air temperature   19.3  Celsius        66.8  Fahrenheit
   [3]Inside air temperature    25.2  Celsius        77.3  Fahrenheit
   [4]Outside relative humidity 88    percent
   [5]Inside relative humidity  49    percent
   [6]Wind direction            322   degrees
   [7]Wind speed                0.4   meter/second   1     miles/hour
   [8]10min wind speed          1.8   meter/second   4     miles/hour
   [9]Air pressure              1003  millibar
   [10]Solar radiation          0     watts/(meter^2)
   [11]UV                       0     UV Index/10
   [12]Rain rate                0     clicks/hour
   [13]Disclaimer
References
   1. https://hpwren.ucsd.edu/Sensors/SDSC/
   2. https://hpwren.ucsd.edu/cgi-bin/Davisgraph.pl?s=198.202.124.3-HPWREN:SDSC:
    Davis:1:0&p=OAT&t=UCSD at SDSC&y=Ou
tside air temperature in degrees Fahrenheit
   3. https://hpwren.ucsd.edu/cgi-bin/Davisgraph.pl?s=198.202.124.3-HPWREN:SDSC:
    Davis:1:0&p=IAT&t=UCSD at SDSC&y=In
side air temperature in degrees Fahrenheit
   4. https://hpwren.ucsd.edu/cgi-bin/Davisgraph.pl?s=198.202.124.3-HPWREN:SDSC:
    Davis:1:0&p=ORH&t=UCSD at SDSC&y=Ou
tside relative humidity in percent
   5. https://hpwren.ucsd.edu/cgi-bin/Davisgraph.pl?s=198.202.124.3-HPWREN:SDSC:
    Davis:1:0&p=IRH&t=UCSD at SDSC&y=In
side relative humidity in percent
   6. https://hpwren.ucsd.edu/cgi-bin/Davisgraph.pl?s=198.202.124.3-HPWREN:SDSC:
    Davis:1:0&p=WD&t=UCSD at SDSC&y=Win
d direction in degrees
```

We do not want that extra text I mentioned because we will begin after this to catalog and parse the data using awk. The way I have it set up, awk would output several items instead of a single item (we would ideally like the data that we are scraping not have extra junk along with it) unless we cut away that excess text. To simply get rid of that extra text we use the sed command with the -i option which allows you to edit files in place. sed is instructed here to look for a string 'References' and then to delete everyting after that string with the $d command. Now that we have cleaned up the data a bit so it is more conducive to our purposes, we will extract all of the useful information from it. This is done simply with the awk command and a separate variable; the awk command is run for every data point that we want to scrape. All of these commands use awk to search the file for a string and then print a field specified by $ and then the number of the field. A field is given here as a string which could be separated from other strings using spaces. So if you had say 4 strings on a line all separated by spaces, the first string would also be the first field, the second string would also be the second field, etc... until the final string being the final field. The field numbers and the strings to search for within the text file are given by the writer of the script, this is not an automated process here. Awk reads the data from the text file generated by lynx and then creates a variable for each piece of data.

Finally, after we have used awk to create our variables, we create a few date variables so we can plot the data easily. A new variable called Date is created using the date builtin and formatting it as dd-mm-yyyy-hh-mm-ss (yyyy is because we used capital Y instead of lower case y, if we had used the lower case we would only get yy for the year formatting). A UNIX time variable is then created by converting the date into seconds using the date builtin command. Next we create a variable (Pacific_Time) which changes the UTC UNIX time to the PDT UNIX time, this is done using the program bc. Here we create a new variable Pacific_Time whose contents are the current UNIX time - 25200 which is the number of seconds difference between UTC and PDT time. This is achieved using bc where we use the scale command to choose a number of significant figures for bc to calculate to and then piping the calculation into bc. Finally, all of these variables are echoed into a string with single spaces separating data points and then appended to the data file we created using the the » redirect. The program then waits 10 min and starts all over again.

Please see the following shell script which is the web scraper in its entirety. If you want to run it yourself, please ensure that lynx is installed on your system as that is the only non-standard package:

```bash
#!/bin/bash
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-: Weather_Scraper.sh -:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

#File="Weather_Scraper.sh"; touch ${File}; chmod +x ${File}; vim ${File}; nohup ./${
    File} &

echo " Checking for lynx and bc packages..."

# check for lynx
lynx_command=`which gnuplot 2>/dev/null`
if [ "$lynx_command" = "" ]; then
        $ECHO
        $ECHO "lynx not in PATH"
        echo "Installing lynx"
  sudo apt-get install lynx
  else
  echo "lynx installed"
fi
# check for bc
bc_command=`which gnuplot 2>/dev/null`
if [ "$bc_command" = "" ]; then
        $ECHO
        $ECHO "bc not in PATH"
        echo "Installing bc"
  sudo apt-get install bc
  else
  echo "bc installed"
fi
# check for gnuplot
gnuplot_command=`which gnuplot 2>/dev/null`
if [ "$gnuplot_command" = "" ]; then
        $ECHO
        $ECHO "gnuplot not in PATH"
        echo "Installing Gnuplot"
  sudo apt-get install gnuplot
  else
  echo "Gnuplot installed"
fi
echo "  done"

sudo timedatectl set-timezone America/Los_Angeles # Set TZ PDT

```

```
Date=$(date '+%d-%m-%Y-%H-%M-%S')  # Set date/time formatting
File_Name=${Date}_UCSD_Weather.dat # Create a dated file name

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:- Create data file :-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

cat > ${File_Name} << EOF
"YYYYMMDD HH:MM:SS Date_Retrieved_%d-%m-%Y-%H-%M-%S Unix_Time Unix_Time->Pacific_Time Temp(C) Temp(F) Humidity(%) Wind_Dir.(deg.) Wind_Speed(mi/hr) Air_Pressure(millibar) UV(Index/10) Rain_Rate(clicks/hr)"
EOF

echo " Writing file ${File_Name}..."; echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-: Create data plotter program :-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

cat > Weather_Plotter.sh << "EOF"
#!/bin/bash

File=${File_Name} # Can be replaced with static file name

Start_Date=$(awk '{if(NR==2) print $5}' ${File})
End_Date=$(awk 'END { print $5 }' ${File})
# Search file for UNIX time where data starts and ends

## Plot Data with Gnuplot
gnuplot <<!
set terminal dumb size 100,50
set xlabel "Date"
set ylabel "Degrees F"
set xdata time
set timefmt "%s"
set format x "%m/%d\n%H:%M"  # sets x-axis formatting \n = newline
set xrange ["$Start_Date":"$End_Date"]   # set xrange [1628825426:1628859039] # Change to start and end points of the Pacific Unix Time # Thanks to Mark Setchell from Stack Overflow
p '${File}' u 5:7 with lines notitle     # Change file name for your needs
!
EOF

echo " Writing file Weather_Plotter.sh..."; echo "  done"
chmod +x Weather_Plotter.sh

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:- Scraping Loop -:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

echo " Scraping from https://hpwren.ucsd.edu/Sensors/SDSC/..."
echo "  Writing to ${File_Name}..."

while [ 1 ] ; do

lynx --dump https://hpwren.ucsd.edu/Sensors/SDSC/ > lynx_dump.txt
# Dump contents of html page into a nicely formatted .txt file
sed -i '/References/,$d' lynx_dump.txt
```

```
# Use sed to delete ($d) all lines after and including the word "References" to make
  awk return only single strings, the as-created lynx_dump.txt file will have
  extra junk at the end which would bother our variable creation with awk
  otherwise


# Parse lynx_dump.txt with awk, use that to create variables:

 Day=$(awk '/San Diego Supercomputer/{print $1}' lynx_dump.txt)
# Make var with awk from field 1, data format: YYYYMMDD
 Time=$(awk '/San Diego Supercomputer/{print $2}' lynx_dump.txt)
# Make var with awk from field 2, data format: HH:MM:SS
 Temp_c=$(awk '/Outside air temperature/{print $4}' lynx_dump.txt)
# Make var with awk from field 4, data format: Celsius
 Temp_f=$(awk '/Outside air temperature/{print $6}' lynx_dump.txt)
# Make var with awk from field 6, data format: Fahrenheit
 Humidity=$(awk '/relative humidity/{print $4}' lynx_dump.txt)
# Make var with awk from field 4, data format: Percent relative
 Wind_Direction=$(awk '/Wind direction/{print $3}' lynx_dump.txt)
# Make var with awk from field 3, data format: Degrees
 Wind_Speed=$(awk '/Wind speed/{print $3}' lynx_dump.txt)
# Make var with awk from field 3, data format: Miles per hour
 Air_Pressure=$(awk '/Air pressure/{print $3}' lynx_dump.txt)
# Make var with awk from field 3, data format: Millibar
 UV=$(awk '/UV/{print $2}' lynx_dump.txt)
# Make var with awk from field 2, data format: UV Index/10
 Rain_Rate=$(awk '/Rain rate/{print $3}' lynx_dump.txt)
# Make var with awk from field 3, data format: Clicks per hour

Date=$(date '+%d-%m-%Y-%H-%M-%S'); Unix_Time=$(date '+%s')
Pacific_Time=$(echo "scale=2;($Unix_Time - 25200)" | bc)

echo $Day $Time $Date $Unix_Time $Pacific_Time $Temp_c $Temp_f $Humidity
  $Wind_Direction $Wind_Speed $Air_Pressure $UV $Rain_Rate >> ${File_Name} #
  Append new data to .dat file

sleep 10m # Site found to update every 1 minute

done
```

A sample of the data file output of this script is given as follows:

```
debian-micro@debian-micro:~$ cat 14-08-2021-06-38-45_UCSD_Weather_Report.dat
"YYYYMMDD HH:MM:SS Date_Retrieved_%d-%m-%Y-%H-%M-%S Unix_Time Unix_Time->
  Pacific_Time Temp(C) Temp(F) Humidity(%) Wind_Dir.(deg.) Wind_Speed(mi/hr)
  Air_Pressure(millibar) UV(Index/10) Rain_Rate(clicks/hr)"
20210814 06:38:19 14-08-2021-06-38-45 1628948325 1628923125 19.2 66.6 100 17 0.0
  1012 0 0
20210814 06:48:38 14-08-2021-06-48-45 1628948925 1628923725 19.4 66.9 100 14 0.0
  1012 0 0
20210814 06:58:07 14-08-2021-06-58-46 1628949526 1628924326 19.4 66.9 100 14 0.0
  1012 0 0
20210814 07:08:26 14-08-2021-07-08-46 1628950126 1628924926 19.6 67.3 100 14 0.0
  1012 0 0
20210814 07:17:54 14-08-2021-07-18-46 1628950726 1628925526 19.9 67.9 100 14 0.0
  1012 0 0
20210814 07:28:25 14-08-2021-07-28-46 1628951326 1628926126 20.2 68.3 100 14 0.0
  1012 0 0
20210814 07:38:44 14-08-2021-07-38-47 1628951927 1628926727 20.2 68.4 100 14 0.0
```

```
    1012 0 0
20210814 07:48:13 14-08-2021-07-48-47 1628952527 1628927327 20.3 68.6 100 85 2.2
    1012 0 0
20210814 07:58:38 14-08-2021-07-58-47 1628953127 1628927927 20.5 68.9 100 59 1.3
    1012 5 0
20210814 08:07:47 14-08-2021-08-08-47 1628953727 1628928527 20.8 69.4 100 59 0.0
    1012 6 0
```

The Gnuplot output in my case looks like the following plot with degrees F on the y-axis and the date in mm/dd with the time below that on the x-axis:

```
debian-micro@debian-micro:~$ ./Weather_Plotter.sh
 76 +------------------------------------------------------------+
    |  *      +      +      +      +      +      +      +      + |
    |  *                                                         |
    |  *                                                         |
    |  *                                          *              |
    |  *                            **           ***             |
    |  *                            **           ***             |
 74 |-+*                            **           ***           +*|
    |  **                           **           ***            *|
    |  **                           **           ***            *|
    |  **                           ***          ***            *|
    | * *                          ** *          ***            *|
    | * *                          *  *          ***            *|
    | * *             *            *  **        ** *            *|
    | * *           *****          *   *        ** *            *|
 72 |-* **          *****          *   *        ** *           +*|
    | * **          *****         *    *        *   *           *|
    |**  **         *****         *    *        *   **          *|
    |**  **         ******        *    *        *   **          *|
    |**  **         * ****        *    *        *    *        ** |
    |**   *         *  ***        *    *        *    *        ** |
    |*    *         *  ***        *    *       **    *        ** |
 70 |*+   *        *    **        *     *      *     *        **-|
    |*    *        *      *       *     *      *     *        ** |
    |*    *        *      *      *      *      *      ** *    ** |
    |*     *       *      *      *      **     *      ******  *  |
    |*     *       *      *      *       **    *          *** *  |
    |*     *       *      *      *        *    *          *  ***  |
    |*     *       *       *     *        **   *            **   |
 68 |-+    *       *        *    *         *  **            ** +-|
    |      **      *        *    *         ** *             **   |
    |       *      *        *    *          ***             **   |
    |       *      *        *** *            *              **   |
    |       **     *         ****                            *   |
    |        **    *          ***                                |
    |         *    *           **                                |
    |         *****                                              |
 66 |-+        ****                                            +-|
    |            **                                              |
    |                                                            |
    |                                                            |
    |                                                            |
    |                                                            |
    |  +      +      +      +      +      +      +      +      + |
  64 +------------------------------------------------------------+
     08/14  08/15  08/15  08/16  08/16  08/17  08/17  08/18  08/18
     12:00  00:00  12:00  00:00  12:00  00:00  12:00  00:00  12:00
                             Date
```

5.4 Use functions to make a user interface video editor script with ffmpeg

The following script was born out of a need to quickly edit a video presentation for a SPIE talk I was recording during the 2020 pandemic for my Ph.D. research. I was having lots of trouble editing the video and encoding it using iMovie and other tools that I had available to me at the time. In the heat of my frustration at not getting things to work properly for some strange file format reason, I Googled as a humorous aside something about how to edit videos in the terminal. What I discovered, fully expecting nothing at all, was that there are really excellent tools for editing video in the shell! One of the most popular of these is called ffmpeg.

I want to give special thanks to the guide from arj.no/2018/05/18/trimvideo/ for help learning about some of the ffmpeg commands, this person's guide was very helpful to me and some of the commands here are inspired by those examples.

This script is used to make modifications to video files. One example of this being useful is if you have a bit of dead air or an unwanted part at the beginning or the end of an audio/video recording. This can be tedious to extract by some means but with the shell it can be a breeze actually. This script follows a similar path to the example where we discuss making a text-based user interface but it purely uses functions that have a switch case and it all runs inside of a while loop. This will become very valuable in section 5.5 because we will show how to create a shell script that can run with specific options (by options I mean things like $ program_name -o -v -...)

```
#!/bin/bash

function selection () {
echo -e "\n"                        # Add a new line with \n
echo -e "Enter your selection \c"   # Suppress a new line with \c
}

function menu () {
clear # Erase the previous input each time this block is run
    echo "======================================"
    echo "========== ffmpeg video editor ==========="
    echo "======================================"
    echo "Enter 1: ffmpeg cuts a movie between two times"
    echo "Enter 2: ffmpeg extracts a fixed duration of a movie"
    echo "Enter q: exit this menu"
    selection

    read answer_one
    case $answer_one in # Start primary switch case block

  1) # Select a portion of a movie to keep and remove the rest
    echo -e "What's the name of the file you'd like to cut?"
    read filename_in  # Read user input and store as $filename_in
    echo -e "What would you like your resulting file to be named?"
    read filename_out # Read user input and store as $filename_out
    echo -e "Specify start and end times next..."
    echo -e "Please give the new start time. hh:mm:ss"
    read time_start   # Read user input and store as time_start
    echo -e "Please give the new end time. hh:mm:ss"
    read time_end     # Read user input and store as time_end
    ffmpeg -i $filename_in -ss $time_start -to $time_end -c:v copy -c:a copy
    $filename_out
    ;;

  2) # Extract a duration of a movie with a given starting time
    echo -e "What's the name of the file you'd like to cut?"
    read filename_in  # Read user input and store as $filename_in1
    echo -e "What would you like your resulting file to be named?"
    read filename_out # Read user input and store as $filename_out1
    echo -e "Specify video start time and video duration next..."
    echo -e "Give desired duration of the resulting video hh:mm:ss"
    read duration      # Read user input and store as duration
    echo -e "Give video cut starting time hh:mm:ss"
    read time_start    # Read user input and store as time_end
    ffmpeg -i $filename_in -ss $time_start -t $duration -c:v copy -c:a copy
    $filename_out
    ;;

  q) exit # This block executes the exit command on the menu
    ;;    # End menu exit command command

    esac  # End primary switch case block
    echo -e "Enter return to continue \c"
    read input_one # New user-input variable called input_one

} # End of menu function

while [ 1 ]; do menu ; done # Runs menu function
```

Obviously ffmpeg can be run on its own without a script, this was just a convenient way to show how to make a menu of options with functions instead of completely with while loops (like in the curses-style text-based user interface example given in section 5.1) at the same time. An example of a properly formatted ffmpeg command could be as follows for a .mp4 formatted video file:

```
ffmpeg -i SPIE_Presentation.mp4 -ss 00:00:00 -to 00:14:00 -c:v copy -c:a copy
    SPIE_Presentation_Trimmed.mp4
```

5.5 Scripting with custom options using getopts, for, if, and while statements, as well as exit and shift conditions

Custom options are something that we see in almost every professionally made shell program that we run in the terminal. One example that we talked about was the options when using tar that we discussed in section 2.2.4. Specifically, we pointed out that running tar without the -v option suppressed its output. This is what options are great for: they give you a choice of whether or not to implement certain features of a program. If you are a curious reader, you may have already wondered up to this point how we might go about adding our own options that can be run with our programs. In this section, I will show you how to do exactly that using a BASH built-in called getopts.

Getopts is a utility that allows you to retrieve arguments about what options you want to have applied to the execution of a script based on a list of parameters that are included in that script. For various reasons, using getopts is vastly easier when you have your program defined as a set of functions like we discussed in section 5.4. That way you can have getopts control and execute your functions within while loops for your various parameters (the options) and have your script default to user-defined functions in the event that no additional options are given.

A large amount of what I have learned about how to use getopts and options came from my reading through the source code of a program for DFT work called WanT. So I would like to thank the team of developers working on the WanT code because a lot of my knowledge of how to use options and getopts comes from my tinkering with and adaptation of some of their code. For anyone interested, the specific program in the WanT code is called run.sh. Additionally, I want to thank kammerl.de/ascii/AsciiSignature.php and its creator for the standard font ASCII art! If you haven't yet, you may consider checking it out for yourself, the developer of that website made it a fantastic resource for creative and fun ASCII art projects that you can incorporate into your own code!

Because I want these chunks of code to be easily readable and also very quickly implementable into your own programs, I am including only a minimally functional main portion of this script and choosing to focus mainly on the use of getopts. This script is similar to the previous in that it is a few menus and sub-menus all defined with functions. Getopts then takes over and controls how the functions behave based on options that are supplied when running the program. The part of this script containing all the functions is highly truncated as well.

One note here is that having a little bit of extra 'breathing room' in your code and plenty of comments is very vital for human-readability and reference when you come back to your code after not having looked at it for a long time. Nevertheless, the header portion of this code is very similar to the script for editing a video file with ffmpeg (see section 5.4) and you should be able to see exactly what its doing without too much hassle.

The following code is divided into three sections. The first is what we just talked about in the previous paragraph and is only minimally functional so that the script isn't too bulky and crowded. The second section gives several variables that we will apply within the getopts. And the third section contains all of the getopts commands as well as what the script should do in the case that no options are given with its execution in the terminal. This third section is, of course, the most pertinent here and will be verbosely commented as well as given more 'breathing room' compared to the first section so that it is more easily human-readable.

This third section does a few interesting things and is the reason we can add options onto a script comprised otherwise of functions and variables. All of the parts of this third section basically tell getopts how to handle using the functions and variables that you have defined in your script. What follows is an explanation of the third section of this script, e.g. everything below the header 'getopts Implementation':

First, several variables are defined: the main variable is called ALLOWED_ACTION that allows options to be called, the other two are dummy variables that we are saving for use in just a bit. Next, we begin the main loop with getopts. Getopts operates within a while loop here and we pass the options that we want to define (here h, m, w, i, and x) via the string :hmwix: (it is very important here that the options you pass to getopts are just single characters and given without any spaces between). The x option is going to operate as a dummy variable that we will store an input into as a variable called $OPTARG. Nothing will happen with the script if we try to run it with just the -x option. The -x option will only function in conjunction with another option written plain words like 'help'. As an example (for the script included below), if we try to run it with the -x option, we will get the following result:

```
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -x
error: x requires an argument
MainUsers-iMac:BASH Programs mainuser$
```

After that, we create a switch/case section defining what actions to take for each of the options we previously passed to getopts; these should be roughly self-explanatory as they do similar things to what I've explained in the text-based user interface section. One exception to the this is use of the 'shift' and 'exit' commands. The shift command (a BASH built-in) more or less is going to remove one or more argument(s) from the beginning of a list based on the argument passed to the shift command. With shift, if we had three variables say $1, $2, and $3 and used shift 2 on them, the following would happen: $1 would be erased, $2 would become $1, and $3 would become $2.

This is handy for reordering options in a list. Shift has many uses most of which are beyond the scope of this text, I recommend that you read up on that subject yourself. However, here we used shift to reorder the list so that if the -x command is given, we will change the way that BASH is interpreting the list of actions and have it consider the three if loops after the primary getopts loop. The exit command (another BASH built-in) allows you to exit the script with a given status, in this case we give that status as 0; this sort of formalism allows you to track what the exit status is for a certain script, e.g. a termination for valid reasons like the script ending, an error, an improper argument, etc... Beyond what the shift and exit commands are doing here, the remainder of the 'while getopts' loop should be more or less self-explanatory.

Next there is a while loop for the $LIST and $ACTION variables that, in the case you do not give any options for getopts to consider, the program will default into normal execution. This while loop has an interesting portion built into the square brackets that allows you to show an indication of why the loop became true. The loop arguments also let us create the variable $ACTION that we use to compare conditional statements of whether the input matches a phrase that we define in our list. If statements control what to do in the event that these comparison conditions are satisfied. These three if statements define the options which can be passed to getopts with full words: these options here are 'help', 'info', and 'menu'. These blocks can be elaborated on in the event that you want to add more full-word options to a script.

Finally, ending the script, we create a dummy variable called $FOUND that we use within a for loop. The for loop and the following if statement control some exit conditions for what is passed to getopts by the user in the variable called $ACTION. Basically, it says to BASH 'in the event that we know what the user is trying to do, based on the list of things that we know how to do, then do the thing, otherwise exit with a pre-defined condition'.

```
#!/bin/bash

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Main Functions :-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

function selection () {
echo -e "\n"; echo -e "Enter your selection \c"
}
function menu () { clear # Erase the previous input each time this block is run
        echo "==========================================="
        echo "============= menu function =============="
        echo "==========================================="
        echo "Enter 1: What happens when you press 1?"
        echo "Enter 2: What happens when you press 2?"
        echo "Enter q to exit this menu"
    selection; read answer_one; case $answer_one in # switch/case
  1) clear; echo "=============== Not very much!! ==========="
            echo "Enter q to exit the menu"
            selection; read answer_two; case $answer_two in
            q) exit ;;
        esac; read input_two ;; # User-defined commands
  2) clear; echo "=========== Still not very much!! ========="
            echo "Enter q to exit the menu"
            selection; read answer_three; case $answer_three in
            q) exit ;;
        esac; read input_three ;;
q) exit ;; # End menu exit command command
esac; echo -e "Enter return to continue \c"; read input_one
```

```
}

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-: Welcome Banner Variables -:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

WELCOME="
                  Welcome!"

WELCOMECONTINUE="
                  Welcome!
           Continue to menu? y/n"

HELPMENU="
 _   _      _         __  __
| | | | ___| |_ __   |  \/  | ___ _ __  _   _
| |_| |/ _ \ | '_ \  | |\/| |/ _ \ '_ \| | | |
|  _  |  __/ | |_) | | |  | |  __/ | | | |_| |
|_| |_|\___|_| .__/  |_|  |_|\___|_| |_|\__,_|
             |_|

       Access the menu with ./menu.sh

    Simple args are -m -i -h
   For info, run ./menu.sh -i
   For menu, run ./menu.sh -m
   For help, run ./menu.sh -h

   More specific args with -x
  For single-use menu, run ./menu.sh -x menu
       For help, run ./menu.sh -x help
       For info, run ./menu.sh -x help"

INFO="
     Menu.sh version 2.0
Created by Steven E. Bopp on March 17, 2019:
For a curses-like text-based user interface
      Materials Science & Engineering"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:- getopts Implementation :-:-:-:-:-:-:-:-:-:
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:

ALLOWED_ACTION="help info menu" # -x * options allowed to be called
ACTION=
LIST=

# Begin getopts primary while loop
while getopts :hmwix: OPT # Define args; *: requires an arg.
do
  case $OPT in
  (x) ACTION="$OPTARG" ; shift 2 ;;
  (h) echo "$HELPMENU" ; exit 0 ;;
  (m) while true;
  do
    menu
  done ;;
  (i) echo "$INFO" ; exit 0 ;;
  (w) clear; echo "$WELCOMECONTINUE" ;
    read welcomecontinue
      case $welcomecontinue in
        y) while true; do menu; done ;; # Run program normally
        n) exit ;; # This navigates the user to the desktop
        q) exit ;;# Exit command from this menu application
      esac # End program launch sub-menu switch case block
    read welcomecontinue ;; # End Welcome (-w)
  (:) echo "error: $OPTARG requires an argument"
      exit 1 ;; # Tells if argument is unknown
  (?) echo "error: unknown option $OPTARG"
      exit 1 ;; # Tells if option is unknown
  esac
done

# Runs menu function in the event that no options are chosen
while [ -z "$LIST" -a -z "$ACTION" ]
do
  menu
done

# Deploy -x * options from ALLOWED_ACTIONS variable help
if [ "$ACTION" = "help" ] ; then
   echo "$HELPMENU"
   exit 0
fi

# Deploy -x * options from ALLOWED_ACTIONS variable info
if [ "$ACTION" = "info" ] ; then
   echo "$INFO"
   exit 0
fi

# Deploy -x * options from ALLOWED_ACTIONS variable menu (one time run since if..fi)
if [ "$ACTION" = "menu" ] ; then
   menu
   exit 0
fi

FOUND=
for allowed in $ALLOWED_ACTION
do
    if [ "$ACTION" = "$allowed" ] ; then FOUND="yes" ; fi
done

if [ -z "$FOUND" ] ; then
   echo "error: unknown action = $ACTION"
   exit 2
fi
```

Some examples of inputs and outputs of this script are as follows:

Here we can see the usefulness of the exit command from before when we try to run the program with –help vs. -help. Exit 2 is given here and prints 'unknown action = $ACTION' to the terminal. This is an example of why using exit with conditions is very handy in lots of scenarios, it allows you to tell yourself why the program halted exactly and then print that to the terminal.

```
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh --help
error: unknown option -
```

```
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -help

 _   _      _         __  __
| | | | ___| |_ __   |  \/  | ___ _ __  _   _
| |_| |/ _ \ | '_ \  | |\/| |/ _ \ '_ \| | | |
|  _  |  __/ | |_) | | |  | |  __/ | | | |_| |
|_| |_|\___|_| .__/  |_|  |_|\___|_| |_|\__,_|
             |_|

       Access the menu with ./menu.sh

    Simple args are -m -i -h
   For info, run ./menu.sh -i
   For menu, run ./menu.sh -m
   For help, run ./menu.sh -h

   More specific args with -x
  For single-use menu, run ./menu.sh -x menu
       For help, run ./menu.sh -x help
       For info, run ./menu.sh -x help
MainUsers-iMac:BASH Programs mainuser$
```

Running the program with the -i or the -info options will give the same results as we've previously defined. This is done for redundancy because evidently there is a split of people who want to use full names for the options and abbreviations for the options. Either one seems valid for me and it seems reasonable to support both possibilities.

```
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -i

     Menu.sh version 2.0
Created by Steven E. Bopp on March 17, 2019:
For a curses-like text-based user interface
      Materials Science & Engineering
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -info

     Menu.sh version 2.0
Created by Steven E. Bopp on March 17, 2019:
For a curses-like text-based user interface
      Materials Science & Engineering
MainUsers-iMac:BASH Programs mainuser$
```

You can also pass the -m option which here I have set to just enter the main portion of the program. This option here is indistinguishable from just running the program without any options.

```
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -m
```

```
==========================================
============= menu function ==============
==========================================
Enter 1: What happens when you press 1?
Enter 2: What happens when you press 2?
Enter q to exit this menu


Enter your selection
```

Another interesting consideration is if we try to pass an option to getopts that we haven't defined. For simplicity, I am calling this unknown command 'command'. Passing an unknown command with the -x option will return an error (which we have defined as 'unknown action') and then reach an exit condition. This is very handy because it allows you to see what exactly has brought you to an exit condition in your script. If we try to just run our script as ./getopts.sh -command, then getopts will only consider the first character following the argument of the hyphen -. Also, we could technically be weird and run with an option like -i_command and, since getopts will only recognize the first character of the options following a hyphen, it will technically follow as if we have run with the -i command.

```
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -x command
error: unkwown action = command
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -command
error: unkwown option u
MainUsers-iMac:BASH Programs mainuser$ ./getopts.sh -i_command

     Menu.sh version 2.0
Created by Steven E. Bopp on March 17, 2019:
For a curses-like text-based user interface
      Materials Science & Engineering
MainUsers-iMac:BASH Programs mainuser$
```

5.6 Automate installation (sort of) with redirects and shell scripts

Another use of shell scripting is to automate the setup of programs and directories, as well as to quickly manipulate the locations of items within a file tree. This can be especially handy when you frequently reinstall or swap operating systems (this may be the case if you just like trying new varieties of Linux, if you're constantly purging your system and fresh-installing bleeding-edge copies of your OS, or maybe you just need to automate the building of file trees on a virtual machine system.

What follows is an admittedly very simple script but it is just intended to show you that you can easily manipulate programs when you are setting up a new system. The program we are going to use as an example is VESTA which I use all the time in my own research. At the time if this writing, it was supplied for Linux as an archive called VESTA-x86_64.tar.bz2

One of the most useful parts of this script I believe is that it shows you how to create new and executable files using cat, a redirect, and then chmod. I use this constantly in my own work to make executables on remote machines and dump code into them using ssh and vim and copy/paste off of my local machine. This functionality is used heavily throughout. The code is mostly self explanatory and just relies on past lessons from this text.

```
#!/bin/bash

# 'Install' VESTA into a certain directory in Ubuntu
  touch VESTA.sh
  echo "cd VESTA/VESTA-x86_64; ./VESTA" > VESTA.sh
  chmod 755 VESTA.sh
  mkdir VESTA
  mv VESTA-x86_64.tar.bz2 VESTA
  cd VESTA
  tar -xvjf VESTA-x86_64.tar.bz2
  rm -r VESTA-x86_64.tar.bz2
```

5.7 Normalize and 'fix' data sets using dos2unix or tr, sed, grep, sort, wc, awk, for, eval, and bc

One common use case of BASH and the shell that I find is to manipulate data sets without the use of a spreadsheet editor. With big sets of data, spreadsheet editors just ends up being cumbersome and slow me down with the tedious copying, pasting, scrolling, and selecting. We can use command line tools to automate the manipulation of data for us so we don't have to bother with the tedium of manual data manipulation! An important caveat is to make sure that the file you are trying to manipulate is compatible with the UNIX system. That is to say, line endings (which the reader can research on their own time as an exercise) vary between the dos, UNIX, and Mac standards. This variation can frustrate implementation of shell-based automatic text editing and manipulation programs like awk if the line endings that they expect to find are not there or if they encounter unknown characters. An example of awk not knowing a certain character is with the dos carriage return character ^M. In the following section we will make a script called x-ray_data_normalizer.sh. At the risk of getting ahead of ourselves a bit, see the output if we run it on a file that is dos-formatted:

```
ubuntu-budgie@ubuntu-budgie:~/Desktop$ ./x-ray_data_normalizer.sh
The maximum intensity is 3026941
There are 800 lines in the datafile
(standard_in) 1: illegal character: ^M
1 0.0050 52007.0
(standard_in) 1: illegal character: ^M
2 0.0150 267845.0
(standard_in) 1: illegal character: ^M
3 0.0250 897316.0
(standard_in) 1: illegal character: ^M
4 0.0350 1855880.0
(standard_in) 1: illegal character: ^M
5 0.0450 2202698.0
(standard_in) 1: illegal character: ^M
6 0.0550 1894395.0
(standard_in) 1: illegal character: ^M
7 0.0650 1827316.0
(standard_in) 1: illegal character: ^M
8 0.0750 1822011.0
(standard_in) 1: illegal character: ^M
9 0.0850 1836177.0
(standard_in) 1: illegal character: ^M
10 0.0950 1842389.0
```

Clearly BASH doesn't like something that's going on here and it turns out it's that pesky ^M carriage return character. If we view part (truncated because there are lots of data points) of the original .csv data file (one made by the X'Pert Epitaxy program on a Windows XP machine), there is no indication that there is anything out of the ordinary:

```
[Measurement conditions]
Sample identification,
Comment,
Anode material,Cu
K-Alpha1 wavelength,1.5405980
K-Alpha2 wavelength,1.5444260
Ratio K-Alpha2/K-Alpha1,0.500
Monochromator used,NO
Generator voltage, 45
```

```
Tube current, 40
File date and time,06-Aug-2021 04:52
Unit cell,
h k l, 0 0 0
Scan axis,2Theta-Omega
Scan range,0.0000,8.0000
Scan step size,0.0100000
Omega offset,0.8530
No. of points, 800
Scan type,CONTINUOUS
Phi,0.0
Psi,0.0
X,0.0
Time per step,2.00
[Scan points]
Angle,Intensity
0.0050,52007.0
0.0150,267845.0
0.0250,897316.0
0.0350,1855880.0
0.0450,2202698.0
0.0550,1894395.0
0.0650,1827316.0
0.0750,1822011.0
0.0850,1836177.0
0.0950,1842389.0
```

This file concludes with the following pesky 0.0 values placed randomly between 1.0 values.

```
MainUsers-iMac:BASH mainuser$ tail *.csv
==> Steven_P4_0720_Gonio_AlN_1_Aug_6_Si_Sub.csv <==
7.9050,1.0
7.9150,0.0
7.9250,0.0
7.9350,0.0
7.9450,0.0
7.9550,0.0
7.9650,1.0
7.9750,1.0
7.9850,0.0
7.9950,0.0

```

The measurement data you are seeing above is from an X-ray reflectometry (XRR) measurement and includes an angle at which the X-rays are shining on a thin film, as well as the intensity of X-rays that are measured by a detector. There are many things about the formatting of this data file that I don't care for and are not useful to me. Examples include all of the header text, the comma data separator character, and the fact that the data set isn't normalized to the maximum intensity (a requirement for a program I use to analyze the data). Additionally, there are some values that are counted as 0.0 near the end. This is significantly frustrating to the use of another program called GenX which I use to analyze this data.

So, to demonstrate how I like to extract all of this data and insert it automatically with only what I want into a new file of the proper UNIX formatting, we will create a script that can do the following: 1) remove unwanted text and labels, 2) change the data file separator from a comma to a space, 3) normalize the data set to the maximum intensity, 4) remove zero values and replace them with the smallest significant value, and 5) change the dos format of the original .csv file to the UNIX format so that awk can operate on it without getting angry and telling us that we're issuing illegal commands.

Please consider the following script (and note that, for technical reasons concerning the sed program, an empty string " needs to be inserted into the beginning of the command if you are trying to run this program on OS X, that empty string should be deleted if you are trying to run this program on UNIX or Linux. We will start by creating three variables, the first of which being the original data set named "Steven_P4_0720_Gonio_AlN_1_Aug_6_Si_Sub.csv" and the second and third being storage variables for when we write the new data set after each step with the tr, awk, sed, and bc commands.

Our first major operation on the data is to create a new file where we will store the properly UNIX-formatted data set. Then we use the tr (translate) command to delete (with the -d option) the carriage return character by redirecting the original dos-formatted file into tr and then redirecting the output into a new file that will contain the properly UNIX formatted output. Thanks to Lee Mendelowitz on Github where the inspiration for this translation came from.

Next I copy the UNIX-formatted file into a new temporary file (since I like to keep the UNIX file unaltered for archival purposes) so that I can perform some text deletion with sed. In the next three four commands I search for and delete all lines before the line containing 'Angle,Intensity' to trim unwanted text. Then I change all instances of 0.0 to 1.0 (the smallest value recorded by the detector that is still usable by an analysis program I use called GenX). After that, I continue using sed to swap all of the commas (which are used as the data point separators) with a space (my preferred data separator). Finally, I use sed to delete the first line of the file which contains the unwanted text 'Angle,Intensity'.

Next, we want to normalize the data set to the maximum intensity. What we do to begin is to search for the maximum value in the entire text file (since the intensity values are much larger than the angle values) using grep. The first part of the grep command is to print numerical values found in the data file into a new temporary file. grep passes the operation to another program called sort where we use -rn for a reverse numerical sort of the temp file's contents, and then we pipe that output file into the program called head (for more on the program called head, see section 2.2.2) where we give it the -n 1 option to just return the top line of the temp file (which will be our largest value of the intensities). Thanks very much to an old post from fedorqui from unix.stackexchange for the inspiration for this command.

After all of that, we are ready to begin setting up our variables for the data normalization. We are going to use a for loop to do the normalization and for that we need a counter variable. That counter variable will be the total number of data points in the data set which we can extract using the wc (word count) command with the -l (number of lines in a file) option and passing that off with a pipe to awk which will cut out unwanted text output that comes with the wc command. We will also create a new variable which tells the script what data point we want to be the beginning of our normalization (the first data point in this case). Finally, before the loop of the main calculation, we print some values to the terminal like the number of lines and the starting data point.

Now we will begin creating our loop to calculate the normalized data set from the original data set: We start a for loop based on all lines in a range of numbers. That range of numbers uses the eval command so that it is pre-computed before the for loop is called and will be a linearly spaced array from the starting line number incremented up by 1 every data point until the number of the last line is hit. That array is echoed into the for loop as a variable and then we start the loop with the do command. At each step of the loop we create two new variables with awk: the first is just pulling the first field from the nth line in the loop, the second is pulling the second field from the nth line in the loop. Next we use a pipe with the bc calculator program to divide the number in the second field of the nth line by the maximum value in the data file (this is the normalization step, we also choose scale=8 in the bc command because we want a lot of significant figures). And finally, at every step, the program will append the new line (consisting of the first field from the original data set and then the second field normalized to the maximum value of the original data set) to an output file.

```bash
#!/bin/bash

Dos_Input_File_Name="Steven_P4_0720_Gonio_AlN_1_Aug_6_Si_Sub.csv"
Output_File_Name="x-ray_data_normalized.dat"

Temp_File_Name="tmp.dat"; touch ${Temp_File_Name}

UNIX_Input_File_Name="UNIX_${Dos_Input_File_Name}"; touch ${UNIX_Input_File_Name}
tr -d '\r' < ${Dos_Input_File_Name} > ${UNIX_Input_File_Name}
# translate dos to UNIX (thanks to Lee Mendelowitz on Github)
#tr '\r' '\n' < file.mac.csv > file.unix.csv
# translate mac to UNIX, replace ^M w/ UNIX line endings
#tr -d '\r' < file.windows.csv > file.unix.csv
# translate dos to UNIX, replace carriage return w/ UNIX endings

cp ${UNIX_Input_File_Name} ${Temp_File_Name}
sed -n '/Angle,Intensity/,$p' ${UNIX_Input_File_Name} > ${Temp_File_Name}
# Search for line containing "Angle,Intensity" and delete everyting before that
sed -i '' 's/,0.0/,1.0/g' ${Temp_File_Name}
# change 0.0 intensity to 1.0 intensity
sed -i '' 's/,/ /g' ${Temp_File_Name}
# add empty string '' in command for OS X (thanks Choon-Chern Lim)
sed -i '' '1d' ${Temp_File_Name}
# remove first line ("Angle,Intensity")

Maximum_Intensity=$(grep -Eo '[0-9]+' ${Temp_File_Name} | sort -rn | head -n 1)
# grep -Eo '[0-9]+' prints all matches of positive decimal integer numbers in the file. Each match will be printed in a different line, as per the -o flag. Sort -rn sorts the list numerically and in reverse, so that the first number is the biggest. Head -n 1 prints the first line (thanks to fedorqui from unix.stackexchange)
echo " The maximum intensity is $Maximum_Intensity"

Number_of_Lines=$(wc -l ${Temp_File_Name} | awk '{ print $1 }')
# wc -l to count number of lines, pipe that into awk to remove the unwanted wc output (Thanks to user Aaron from askubuntu.com)
echo " There are $Number_of_Lines lines in the datafile"

Starting_Line_Number=1

echo " Writing normalized data into ${Output_File_Name} starting at line ${Starting_Line_Number}..."

for line in $(eval echo "{$Starting_Line_Number..$Number_of_Lines}"); do
# eval builtin concatenates arguments into one single command (thanks to Vivek Gite from cyberciti.biz)

  Column_1=$(awk -v line="$line" '{if(NR==line) print $1}' ${Temp_File_Name})
  # awk -v inserts variable in a way that's legal to the awk command (Thanks to Jotne from StackOverflow)
  Column_2=$(awk -v line="$line" '{if(NR==line) print $2}' ${Temp_File_Name})

  Column_2_Normalized=$(echo "scale=8;(($Column_2)/($Maximum_Intensity))" | bc)

  #echo $line $Column_1 $Column_2 $Column_2_Normalized

  echo $Column_1 $Column_2_Normalized >> $Output_File_Name

done
```

We can run the above program with the following command and view its output (assuming that you name your program the same as mine):

```
MainUsers-iMac:BASH mainuser$ ./x-ray_data_normalizer.sh
 The maximum intensity is 3026941
 There are 800 lines in the datafile
 Writing normalized data into x-ray_data_normalized.dat starting at line 1...
```

The new data file generated by the script can be easily viewed with cat but that might be tedious since it contains 800 lines (and that's a relatively short data set compared to many that I use). An alternative way to view the beginning and end of the data set is with the head and tail commands. Please see their use below in viewing the first ten and final ten lines of the newly normalized data file:

```
MainUsers-iMac:BASH mainuser$ head x-ray_data_normalized.dat
0.0050 .01718137
0.0150 .08848702
0.0250 .29644317
0.0350 .61312063
0.0450 .72769769
0.0550 .62584470
0.0650 .60368404
0.0750 .60193145
0.0850 .60661142
0.0950 .60866366
```

Additionally, if we look at the end of the new, normalized data file, we will see that we have none of those pesky 0 values which would have bothered us before.

```
MainUsers-iMac:BASH mainuser$ tail x-ray_data_normalized.dat
7.9050 .00000033
7.9150 .00000033
7.9250 .00000033
7.9350 .00000033
7.9450 .00000033
7.9550 .00000033
7.9650 .00000033
7.9750 .00000033
7.9850 .00000033
7.9950 .00000033
```

I'd say that, every time I have to do X-ray measurements like these I need to normalized the data set and remove pesky values like the 0.0s as well as trim unwanted text and (just for making my like easier with Gnuplot) change the data separator characters from a comma to a space. Whenever I do that manually with a spreadsheet editor program it maybe takes about five to ten minutes because of fiddling with formatting, special copy and paste functions, and export settings. So doing that several hundred times would have been a significant bore that can be saved with the little BASH scripting tool that I outlined above!

To answer a question that may or may not be brewing in the mind of the more code veteran reader: yes, of course there are many other ways of manipulating data automatically, this just fits with my workflow, machines, and aesthetic as well and fits well into the scope of this text.

5.8 Convert obscure .xrdml files to two-column .dat with tr, awk, paste, bc, and wc

As has been mentioned in this text, there are many times when a scientist will be using a legacy measurement system or something similar and, due to the sensitive nature of those systems to software updates, the computer and its software may be severely outdated. Additionally, it may be the case that data are stored on obscure file formats or that the data might conventionally use a file converter program that is supplied by a vendor with the measurement instrument. Unfortunately, it is not always easy to convert these specific formats to ones that are more compatible with programs like GNUplot, Octave, etc... without the use of the original software. However, it can sometimes pay off to do some of your own searching into how the data are stored within that file format.

One such example is the .xrdml file format that is used with some X-ray measurement systems. An example of one such output from a machine that's from around the year 2002 is included (with redacted information specific to the machine) in the following .xml formatted code (bear in mind that this is a nonsense measurement and does not represent original or valuable research):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xrdMeasurements xmlns="http://www.xrdml.com/XRDMeasurement/1.0" xmlns:xsi="http://
    www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.xrdml.com/
    XRDMeasurement/1.0 http://www.xrdml.com/XRDMeasurement/1.0/XRDMeasurement.xsd"
    status="Completed">
  <sample type="To be analyzed">
    <id></id>
  </sample>
  <xrdMeasurement measurementType="Scan" status="Completed">
    <comment>
      <entry>2Theta-Omega w/offset</entry>
    </comment>
    <usedWavelength intended="K-Alpha 1">
      <kAlpha1 unit="Angstrom">1.5405980</kAlpha1>
      <kAlpha2 unit="Angstrom">1.5444260</kAlpha2>
      <kBeta unit="Angstrom">1.3922500</kBeta>
      <ratioKAlpha2KAlpha1>0.5000</ratioKAlpha2KAlpha1>
    </usedWavelength>
    <incidentBeamPath>
      <radius unit="mm">200.00</radius>
      <xRayTube id="xxxxxxx" name="xxxxxx/00 Cu LFF xxxxxxxx">
        <tension unit="kV">45</tension>
        <current unit="mA">40</current>
        <anodeMaterial>Cu</anodeMaterial>
        <focus type="Point">
          <length unit="mm">12.0</length>
          <width unit="mm">0.4</width>
          <takeOffAngle unit="deg">6.0</takeOffAngle>
        </focus>
      </xRayTube>
    </incidentBeamPath>
    <diffractedBeamPath>
      <radius unit="mm">200.00</radius>
      <detector id="xxxxxxx" name="xxxxxx/xx (Miniprop. large window)" xsi:type="pointDetectorType">
        <phd>
          <lowerLevel unit="%">35.0</lowerLevel>
          <upperLevel unit="%">80.0</upperLevel>
        </phd>
      </detector>
    </diffractedBeamPath>
    <scan appendNumber="0" mode="Continuous" scanAxis="2Theta-Omega" status="Completed">
      <header>
        <startTimeStamp>xxxxxxxxxxxxxxxxxxxxxxx</startTimeStamp>
        <endTimeStamp>xxxxxxxxxxxxxxxxxxxxxxxxxxx</endTimeStamp>
        <author>
          <name>XRay</name>
        </author>
        <source>
          <applicationSoftware version="2.0d">X'Pert Data Collector</applicationSoftware>
          <instrumentControlSoftware version="2.8">XPERT-MPD</instrumentControlSoftware>
          <instrumentID>xxxxxxxxxxxxxxxxx</instrumentID>
        </source>
      </header>
      <dataPoints>
        <positions axis="2Theta" unit="deg">
          <startPosition>44.010</startPosition>
          <endPosition>46.010</endPosition>
        </positions>
        <positions axis="Omega" unit="deg">
          <startPosition>22.0050</startPosition>
          <endPosition>23.0050</endPosition>
        </positions>
        <positions axis="Phi" unit="deg">
          <commonPosition>0.0</commonPosition>
        </positions>
        <positions axis="Psi" unit="deg">
          <commonPosition>0.0</commonPosition>
        </positions>
        <positions axis="X" unit="mm">
          <commonPosition>0.0</commonPosition>
        </positions>
        <commonCountingTime unit="seconds">2.00</commonCountingTime>
        <intensities unit="counts">216 191 205 183 211 201 199 189 204 181 184 187 192 196 209 208 203 174 181 221 202 208 200 209 200 188 208 186 192 191 212 195 175 201 193 192 182 193 197 195 197 202 199 201 192 173 185 188 202 173 192 222 203 191 181 198 189 192 183 183 189 188 203 192 176 193 204 192 176 181 187 198 176 185 191 203 172 183 178 201 169 207 155 196 207 190 183 171 172 199 203 167 182 205 187 182 167 202 184 179 169</intensities>
      </dataPoints>
    </scan>
  </xrdMeasurement>
</xrdMeasurements>
```

A careful observer may detect that there are data points held in the .xml file between the <intensities> xml tags and that there are start and end positions for the 2θ values included in the <startPosition> and <endPosition> tags. That's something we can work with! Hooray! (Also thanks to the original developers of this file format because, from the original documentation of the format, it looks like they tried to make it readable and accessible without too much hassle or obfuscation).

Included below is a highly annotated script that is intended to extract the intensity and 2θ information out of this .xml file and then write it to a two-column data file that can be used much more easily and without complicated file converter programs for plotting and reading of data. The script is based heavily on the data normalization script given in section 5.7 so some of the introductory parts of the explanation will be skipped.

First, like in the data normalizer script, variables for file names are given and some empty data files are created. Next, a pair of awk commands are used to extract the lines pertaining to the 2θ start and end positions as well as the block of text holding the intensity information into a new file called tmp. From the tmp file, the actual 2θ start and 2θ end positions as well as the string of just intensity values are copied into variables $a, $b, and $c. Additionally, the wc command is used to count the number of data points in the intensity string and store that value as $d.

For plotting, we need two vectors of equal size (e.g. two strings of numbers each having equal numbers of elements in the respective strings). To make a new vector that contains the angle information, we need to know the 2θ spacing between the values that are given in the intensity string. This value is calculated by simply subtracting the end angle from the start angle and dividing that value by the number of data points that are in the intensity string. That math is done with bc and stored in a variable called $step_size. Of key importance is that the number of places after the decimal is large enough so as to not create round-off errors that would lead to a miscount in the number of individual steps in the 2θ string. We can ward ourselves against this by choosing a large number for the scale in bc like 20 (in some instances of using this script, a scale of 6 can be too small and lead to a loss of a data point causing inaccuracies in plotting so choose a large scale value).

Next, we use the str command to make a new string for the 2θ values corresponding to the intensity values between a 2θ start and a 2θ end value with the $step_size variable we just calculated. Wrapping up, we re-write the tmp file first with the 2θ values and use the translate command to transpose that string (which is currently a row vector) into a column vector and store that column vector in a new temporary file. This is done by transposing the data set with the tr command. The same treatment is given to the intensity information.

To complete our data file with two column vectors, we use the paste command to combine the individual column vectors from the two temporary files that we just created into a single data file with two columns that have a single space for the data separator. We also add a title to each of the columns of "2Theta" and "Intensity" for ease of reference. Finally, we delete unnecessary temporary files and can use whatever plotting engine we so desire to make our data plots.

Please see the script included below:

```bash
#!/bin/bash

xrdml_Input_File_Name="data.xrdml"
Output_File_Name="${xrdml_Input_File_Name}_converted.dat"; touch ${Output_File_Name}

Temp_File_Name="tmp"; touch ${Temp_File_Name}
#Scan_Parameters_File_Name="Scan_Parameters.txt"; touch ${Scan_Parameters_File_Name}

UNIX_Input_File_Name="UNIX_${xrdml_Input_File_Name}"; touch ${UNIX_Input_File_Name}
tr -d '\r' < ${xrdml_Input_File_Name} > ${UNIX_Input_File_Name}
# translate dos to UNIX (thanks to Lee Mendelowitz on Github)
#tr '\r' '\n' < file.mac.csv > file.unix.csv
```

```
# translate mac to UNIX, replace ^M w/ UNIX line endings
#tr -d '\r' < file.windows.csv > file.unix.csv
# translate dos to UNIX, replace carriage return w/ UNIX endings

awk '/<positions axis="2Theta" unit="deg">/{x=NR+2;next}(NR<=x){print}' ${UNIX_Input_File_Name} >> ${Temp_File_Name}
# Print two lines after the string <positions axis="2Theta" unit="deg"> (thanks to Guru Prasad from UNIX School)
awk '/<intensities unit="counts">/{print}' ${UNIX_Input_File_Name} >> ${Temp_File_Name}
# Print two lines after the string <positions axis="2Theta" unit="deg"> (thanks to Guru Prasad from UNIX School)

a=$(awk 'NR==1{print $1}' ${Temp_File_Name}) # a is a dummy variable to hold 2Theta data
start=$(echo $a | cut -c16- | rev | cut -c17- | rev); # echo $start
# Retrieves the starting 2Theta angle and extracts it from the surrounding xml text with rev and cut
b=$(awk 'NR==2{print $1}' ${Temp_File_Name}) # b is a dummy variable to hold 2Theta data
end=$(echo $b | cut -c14- | rev | cut -c15- | rev) ; # echo $end
# Retrieves the ending 2Theta angle and extracts it from the surrounding xml text with rev and cut
c=$(awk 'NR==3' ${Temp_File_Name}) # c is a dummy variable to hold intensity data
intensities=$(echo $c | cut -c28- | rev | cut -c15- | rev) ; # echo $intensities
# Retrieves the intensity data and extracts it from the surrounding xml text with rev and cut
d=$(echo $intensities | wc -w) # d is a dummy variable to hold the number of intensity data points
# Counts the number of data points stored in the intensity string

echo "2Theta range from $start to $end degrees with $d data points"

step_size=$(echo "scale=20;(($end)-($start))/($d)" | bc); echo "Step size is $step_size"
# Use the bc calculator to calculate a step size based on the difference from the start to the end angle and the numbre of intensity data points. The large value of scale is super important to not have rounding errors that cause a miscount of points.
two_theta=$(seq -w $start $step_size $end); e=$(echo $two_theta | wc -w) # e is a dummy variable to hold the number of created 2Theta data points for comparison with the number of inteisty data points (they should be equal, if not somehting's gone terribly wrong)
# Create a sequence corresponding to the 2Theta values evenly spaced based on the step size variable we just created

echo "Created step size string with $e steps for $d data points"

echo $two_theta > ${Temp_File_Name} # Re-write temp file with the two_theta data
tr ' ' '\n' < tmp > tmp_two_theta_transposed # Use tr command to transpose the data and then write it into a temp file
echo $intensities > ${Temp_File_Name} # Re-write temp file with the intensity data
tr ' ' '\n' < tmp > tmp_intensities_transposed # Use tr command to transpose the data and then write it into a temp file

echo '"2Theta" "Intensity"' > ${Output_File_Name}

paste -d" " tmp_two_theta_transposed tmp_intensities_transposed >> ${
    Output_File_Name}
# Use the paste command to combine the transposed data sets into one file with two
    columns with a single space separating them

rm ${Temp_File_Name}; rm tmp_two_theta_transposed; rm tmp_intensities_transposed #
    clean up files
```

An example terminal output of this script may be the following:

```
MainUsers-iMac:Desktop mainuser$ ./xrdml_converter.sh
2Theta range from 44.010 to 46.010 degrees with      101 data points
Step size is .01980198019801980198
Created step size string with      102 steps for      101 data points
MainUsers-iMac:Desktop mainuser$
```

And the head of a resulting output file may be the following:

```
MainUsers-iMac:Desktop mainuser$ head data.xrdml_converted.dat
"2Theta" "Intensity"
44.010000 216
44.029802 191
44.049604 205
44.069406 183
44.089208 211
44.109010 201
44.128812 199
44.148614 189
44.168416 204
MainUsers-iMac:Desktop mainuser$
```

6 Shell Scripts for DFT Calculations with Quantum ESPRESSO (PWscf)

Quantum Espresso (QE) [2, 3] is an open-source and fully-featured code for DFT, molecular dynamics, and more all for absolutely free (I was going to say 'no charge' here but someone would definitely made a joke haha). For all its capabilities, Quantum Espresso has its quirks (at least at the time of this writing) and I found it to have a steep learning curve when I was just starting out. Here, to alleviate some of that learning curve, I intend to supply some of my scripts for whoever may need them as a reference for their own calculations.

In Quantum Espresso, unlike in VASP and some others, you need to very carefully supply each and every parameter for every calculation in every step. This is done with a variety of executables that you run for different parts of your calculations and with different parts of the input files which include sections like &control, &system, and &electrons. Plotting of the data is most frequently done in GNUplot; for a refresher on that program, please refer to section 3.8.

That being said, as with all of my other code supplied in this book or elsewhere, everything is supplied with absolutely zero guarantee or warranty. Even though every effort is made to have code supplied here work out of the box, some things here and there may require tinkering on the part of the user to get working as they intend or as new developments or releases in DFT packages are introduced. However, the value that I see in supplying this code is to reduce the overall activation energy (see, haha I can make jokes!) for someone to get started with calculations of this type.

Included below are several example scripts as well as usage of supporting programs called Cif2Cell [4] and Firmi which can be used with Quantum Espresso to make the reader's life just a bit easier. In most cases, I owe huge thanks to the people who have created the example scripts that some of these files are based around. I also make attempts to explain potential error sources and some ways of correcting those errors. However, all burden is placed squarely on the reader for verifying the quality of their calculations.

Also, a quick word on pseudopotentials in general. There are many that are supplied by many organizations or individuals. For Quantum Espresso there are some that are supplied on their website. However, it seems to be consensus from members of the community that the Vanderbilt pseudopotentials are very trustworthy and may be a good replacement for those supplied by Quantum Espresso. Additionally, in my experience, some pseudopotentials that I have downloaded did not come readable by Quantum Espresso because the had some extraneous characters at the header of the file(s). Be aware of this, my recommendation is to check the text of all of your pseudopotential files before you use them. Additionally, it is wise to seek verification that the specific potential files that you are using do actually give an accurate representation of reality.

6.1 Compiling Quantum Espresso

Compiling Quantum Espresso should be fairly straightforward just based on the instructions included with their code distributions. However, one stumbling block that I encountered when trying to compile their code on my Ubuntu system was with the installation of Open MPI so that I could parallelize the calculations.

At the time of this writing, installation of OpenMPI can be done using the following commands on the vanilla release of Ubuntu:

```
sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev
```

Downloading Quantum Espresso is also straightforward and can be done easily with wget or curl (as is described in section 3.6) directly from the terminal. Running a parallel computation with quantum espresso can be done (as a general template) with the following (after setting up all of the proper environment variables and things like they tell you do do in the documentation):

Run a parallel calculation:

```
mpirun '/home/qe/bin/pw.x' -in scf.in > scf.out
```

As repositories in Linux are not the same over all time, and Quantum Espresso packages may vary slightly from one distribution to the next, the code supplied above or as follows in this section may require some modification on the part of the reader to suit their specific systems or to reflect the current way(s) that Open MPI should be installed on their system(s).

6.2 Some common error sources in Quantum ESPRESSO

I usually encounter all sorts of errors when I run calculations, especially when I am trying a new-to-me code. Here is a partial list of some error sources and potential solutions that I have found for use with Quantum Espresso. I am including these items with the presentation first of the error you may see, and then a way you may attempt to resolve the error.

Error	Potential Resolution
NSCF calculation halts unexpectedly with an error like "Error condition encountered during test: exit status = 1 Aborting"	try increasing nbnd to a larger number
.out file returns an error with 'Bad Fermi Energy'	try to increase the value of nbnd to a larger number and that may resolve the error (do this sparingly though and try to not stray from a reasonable number for nbnd)
Gnuplot misbehaves in instances where nx and ny values have been changed from their defaults	check that the nx, and ny are not the culprit of any misbehavior before worrying that there is some catastrophic error with the script
epsilon.x returns an error regarding division by zero	there is likely an issue with the intersmear parameter being too small which leads to strange infinities

6.3 Quantum Espresso file headers and environment variables

Because I'd prefer to not include the header file (for the sake of space and redundancy) in every one of the following scripts, and since I have done some tinkering with the layout and items included in the headers, I will include it here as a separate file. Specifically, for the sake of brevity and convenience, I am including the header file for the script in section 6.6 directly following this exposition. The epsilon.x script, which we will discuss more in section 6.6, is intended to calculating the complex dielectric function of Au. We will see however in that section, there are many more considerations which need to be taken into account to get an accurate calculation of permittivity from first principles.

Overall, the below script is separated into several parts. The first is a simple header which reminds the user (in a similar way to what I describe in sections 12.3, and 12.2) of several things like how it is necessary to have GNUplot installed and to make sure that all of the pseudopotentials are readable by the system. Next, sed is used to extract the name of the path and the test command is issued to see if we can use the echo -e option. This is a nice part of the script because it also accounts for the case where the echo -e option is not present and, in that case, sets ECHO as a variable with options for the positive and negative cases. I thought that was cool!

Next, we set up the environment variables path which will be specific to each user. The environment variables path should link the script to the directory containing all of the quantum espresso binaries as well as the directory containing all of your pseudopotentials. The program checks to see if GNUplot is installed in the system (this chunk is actually the exact inspiration and credit for where I describe a use case of the if command in section 4.3.4). All of the normal directories and binaries are also checked in the same way, as well as the pseudopotentials. The pseudopotential checking block also has a clever (as designed by the Quantum Espresso developers so all credit to them) functionality that, in the case it does not find the potential you are directing it to look for, it will automatically download it from their pseudopotential repository and place it into the correct directory.

The final chunk of this script is a list of executable locations. Because I really don't like waste time re-doing work that can be automatically performed when I can avoid it, I prefer to include links and variables relating to all of the executables that I generally use within every script. Another benefit of this is that it's one less thing you need to worry about checking in your script every time that you run it (it's wise to eliminate as many sources of silly errors as is reasonably feasible). All of these executables are stored as variables which can be called from within the script. Finally, the script concludes with the start of a timer program that I find useful for benchmarking CPU hours spent on the individual parts of a calculation.

```
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:Begin Environment Directions-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-Make Certain That All Pseudopotentials are Readable-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-and That all Executables are Called Below-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-Make Certain That All Static Directory Links are Correct:-:-:-:-:-
#-:-:-:-:-:-:-:Install GNUPlot and Inkscape to Have Full Functionality:-:-:-:-:-
#-:-:-:-:-:-:Make that the shell is set to BASH, remove timer otherwise-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# run from directory where this script is
cd `echo $0 | sed 's/\(.*\)\/.*/\1/'` # extract path name
EXAMPLE_DIR=`pwd`

# check whether echo has the -e option
if test "`echo -e`" = "-e" ; then ECHO=echo ; else ECHO="echo -e" ; fi

$ECHO
$ECHO "$EXAMPLE_DIR : starting"
$ECHO
$ECHO "This code shows how to use epsilon.x to calculate the permittivity of Au"
$ECHO "Permittivity is calculated after SCF and NSCF calculations"
$ECHO "and then plotted with literature values using Gnuplot"

# set the needed environment variables
. /home/uname/Documents/Quantum_Espresso/qe-6.0/environment_variables
```

```

# required executables and pseudopotentials
BIN_LIST="pw.x pp.x plotrho.x bands.x plotband.x dos.x projwfc.x fs.x epsilon.x"
PSEUDO_LIST="Au.rel-pbesol-n-nc.UPF"

$ECHO
$ECHO "  executables directory: $BIN_DIR"
$ECHO "  pseudo directory:      $PSEUDO_DIR"
$ECHO "  temporary directory:   $TMP_DIR"
$ECHO
$ECHO "  checking that needed directories and files exist...\c"

# check for gnuplot
GP_COMMAND=`which gnuplot 2>/dev/null`
if [ "$GP_COMMAND" = "" ]; then
        $ECHO
        $ECHO "gnuplot not in PATH"
        $ECHO "Results will not be plotted"
fi

# check for directories
for DIR in "$BIN_DIR" "$PSEUDO_DIR" ; do
    if test ! -d $DIR ; then
        $ECHO
        $ECHO "ERROR: $DIR not existent or not a directory"
        $ECHO "Aborting"
        exit 1
    fi
done
for DIR in "$TMP_DIR" "$EXAMPLE_DIR/results" ; do
    if test ! -d $DIR ; then
        mkdir $DIR
    fi
done
cd $EXAMPLE_DIR/results

# check for executables
for FILE in $BIN_LIST ; do
    if test ! -x $BIN_DIR/$FILE ; then
        $ECHO
        $ECHO "ERROR: $BIN_DIR/$FILE not existent or not executable"
        $ECHO "Aborting"
        exit 1
    fi
done

# check for pseudopotentials
for FILE in $PSEUDO_LIST ; do
    if test ! -r $PSEUDO_DIR/$FILE ; then
       $ECHO
       $ECHO "Downloading $FILE to $PSEUDO_DIR...\c"
            $WGET $PSEUDO_DIR/$FILE $NETWORK_PSEUDO/$FILE 2> /dev/null
    fi
    if test $? != 0; then
        $ECHO
        $ECHO "ERROR: $PSEUDO_DIR/$FILE not existent or not readable"
        $ECHO "Aborting"
        exit 1
    fi
done
$ECHO " done"

# Executable locations, run instructions (stored as variables), and terminal text to
    be returned when run
       FS_COMMAND="$BIN_DIR/fs.x "
       PW_COMMAND="$PARA_PREFIX $BIN_DIR/pw.x $PARA_POSTFIX"
       PP_COMMAND="$PARA_PREFIX $BIN_DIR/pp.x $PARA_POSTFIX"
      DOS_COMMAND="$PARA_PREFIX $BIN_DIR/dos.x $PARA_POSTFIX"
      EPS_COMMAND="$PARA_PREFIX $BIN_DIR/epsilon.x $PARA_POSTFIX"
    BANDS_COMMAND="$PARA_PREFIX $BIN_DIR/bands.x $PARA_POSTFIX"
  PROJWFC_COMMAND="$PARA_PREFIX $BIN_DIR/projwfc.x $PARA_POSTFIX"
  PLOTRHO_COMMAND="$BIN_DIR/plotrho.x"
 PLOTBAND_COMMAND="$BIN_DIR/plotband.x"
$ECHO
$ECHO "  running pw.x as:       $PW_COMMAND"
$ECHO "  running pp.x as:       $PP_COMMAND"
$ECHO "  running fs.x as:       $FS_COMMAND"
$ECHO "  running dos.x as:      $DOS_COMMAND"
$ECHO "  running bands.x as:    $BANDS_COMMAND"
$ECHO "  running gnuplot as:    $GP_COMMAND"
$ECHO "  running epsilon.x as:  $EPS_COMMAND"
$ECHO "  running plotrho.x as:  $PLOTRHO_COMMAND"
$ECHO "  running projwfc.x as:  $PROJWFC_COMMAND"
$ECHO "  running plotband.x as: $PLOTBAND_COMMAND"
$ECHO

START_TIME=$SECONDS # Begin elapsed time measurement (thanks Tom Anderson from
    StackOverflow)
```

Additionally, I am including in the following a sample of the environment_variables file that I used for a long time in one of my own installs of Quantum Espresso. I think that including this, while it is perhaps not as important to the experienced user, would have been helpful to me when I was just starting out. For that reason, I am including it briefly.

As a reader can see, there are very few changes that need to be made compared to the as-supplied environment_variables file. All I did was to change the PREFIX, BIN_DIR, PSEUDO_DIR, and TMP_DIR locations based on where they were located in my specific system. If you are having trouble with determining the location of a specific directory to give to Quantum Espresso, then at any time in the command prompt you can type 'pwd' and find the full path to that directory. This command is very handy for saving time with copying and pasting full file paths directly from the terminal into a script.

Please see the following environment_variables file below:

```
# environment_variables -- settings for running Quantum ESPRESSO examples

LC_ALL=C
export LC_ALL

######## YOU MAY NEED TO EDIT THIS FILE TO MATCH YOUR CONFIGURATION ########

# BIN_DIR = path of compiled executables
#     Usually this is $PREFIX/bin, where $PREFIX is the root of the
#     Quantum ESPRESSO source tree.
# PSEUDO_DIR = path of pseudopotentials required by the examples
#     if required pseudopotentials are not found in $PSEUDO_DIR,
#     example scripts will try to download them from NETWORK_PSEUDO
# TMP_DIR = temporary directory to be used by the examples
#     Make sure that it is writable by you and that it doesn't contain
#     any valuable data (EVERYTHING THERE WILL BE DESTROYED)

# The following should be good for most cases

PREFIX=`/home/steven/Documents/qe-6.4.1/ ; pwd`
BIN_DIR=/home/steven/Documents/qe-6.4.1//bin
PSEUDO_DIR=~/Documents/Quantum_Espresso/qe-6.0/pseudo
# Beware: everything in $TMP_DIR will be destroyed !
TMP_DIR=/home/steven/Documents/qe-6.4.1//tempdir

# There should be no need to change anything below this line

NETWORK_PSEUDO=http://www.quantum-espresso.org/wp-content/uploads/upf_files/


# wget or curl needed if some PP has to be downloaded from web site
# script wizard will surely find a better way to find what is available
if test "`which curl`" = "" ; then
   if test "`which wget`" = "" ; then
      echo "wget or curl not found: will not be able to download missing PP"
   else
      WGET="wget -O"
      # echo "wget found"
   fi
else
   WGET="curl -o"
   # echo "curl found"
fi

# To run the ESPRESSO programs on a parallel machine, you may have to
# add the appropriate commands (poe, mpirun, mpprun...) and/or options
# (specifying number of processors, pools...) before and after the
# executable's name.  That depends on how your machine is configured.
# For example on an IBM SP4:
#
#     poe             pw.x -procs 4              < file.in > file.out
#     ^^^ PARA_PREFIX      ^^^^^^^^ PARA_POSTFIX
#
# To run on a single processor, you can usually leave them empty.
# BEWARE: most tests and examples are devised to be run serially or on
# a small number of processors; do not use tests and examples to benchmark
# parallelism, do not run on too many processors

PARA_PREFIX=" "
PARA_PREFIX="mpirun -np 4"
#
# available flags:
#                  -ni n        number of images        (or -nimage)
#                               (only for NEB; for PHonon, see below)
#                  -nk n        number of pools         (or -npool, -npools)
#                  -nb n        number of band groups   (or -nbgrp,-nband_group)
#                  -nt n        number of task groups   (or -ntg, -ntask_groups)
#                  -nd n        number of processors for linear algebra
#                                             (or -ndiag, -northo)
#
```

```
PARA_POSTFIX=" -nk 1 -nd 1 -nb 1 -nt 1 "
#
# The following variables are used for image parallelization of PHonon
# (see example in PHonon/examples/Image_example)
# NB: the number of processors in PARA_IMAGE_PREFIX is the product of the
# number of processors in PARA_PREFIX and the number of images in
# PARA_IMAGE_POSTFIX
#
PARA_IMAGE_POSTFIX="-ni 2 $PARA_POSTFIX"
PARA_IMAGE_PREFIX="mpirun -np 4"

# function to test the exit status of a job
check_failure () {
    # usage: check_failure $?
    if test $1 != 0
    then
        echo "Error condition encountered during test: exit status = $1"
        echo "Aborting"
        exit 1
    fi
}
```

6.4 Charge density and the electron localization Function (ELF)

Quantum Espresso (QE) can be used to calculate many different properties of materials, two of which being the charge density and the electron localization function (ELF) over a given surface in the unit cell. The charge density is self explanatory and the ELF more or less gives an explanation of the spatial extent over which an electron is localized within a crystal. These properties are very useful in that the ELF gives a visual basis to pair with intuition for the understanding of some of the electronic properties of a crystal. Additionally, and to an admittedly somewhat shallow extent, it an also be used to make very appealing-looking scientific visualizations. Below is the body (since the header is left out and explained in section 6.3) of a Quantum Espresso input script for calculating the change density and the ELF in a TiN crystal.

The following script is separated into eleven separate sections, reader don't beware however because many of these are either repetitive or very easy to understand (at least at the surface level) and will be explained section-by-section but briefly in the following paragraphs.

The script begins with the implementation of a timer so that the user can see a timestamp printed to the terminal at every time that a major portion of the calculation has concluded. The first major section of this script begins immediately after the timer is begun and is used to calculate the self consistent field (SCF) for the TiN crystal that we are considering. We give the familiar input parameters for QE and tell the program to calculate in the following order: $\Gamma \to X \to W \to K \to L \to \Gamma$. The crystal is given in terms of a cell dimension and the fractional atomic positions of the two basis atoms: one for Ti, and one for N (since the rocksalt structure of TiN can be thought of as roughly two interleaved simple cubic structures where the corner of one cube is located at the center of the other cube and all faces of the interleaved cubes are orthogonal or parallel to each other.

Section two uses the code PP.x (Post-Processing FORTRAN program) to compute the charge density using the SCF output from above as its own input. There are several notes for running this calculation which follow here: Make sure that the vectors e1 and e2, which the user must define for QE, are orthogonal. Make sure that the isovalue is large enough for your calculations and also make sure to set x0(...) at the origin of your crystal system. Setting iflag = 2 will yield data for latter creating a 2D plot. The output format setting can be changed based on which program you want to be able to natively read the data: 3 will be formatted for xcrysden, and 2 will be natively formatted for plotrho. The option nfile will tell QE the number of data files that you want to read, and plot_num = 0 is the option to set the electron (pseudo-) charge density.

In section three, we begin using plotrho (which is part of PostProc and produces PostScript 2D plots) to create some plots. The first argument in this section is the input file defined by fileout=... in the PP.x code. The second argument is the output file name that you want to assign to the newly created file. The third argument gives yes (y) or no (n) for logarithmic scaling on the plot. And, finally, the fourth argument gives rho_min rho_max and the number of contour plot levels.

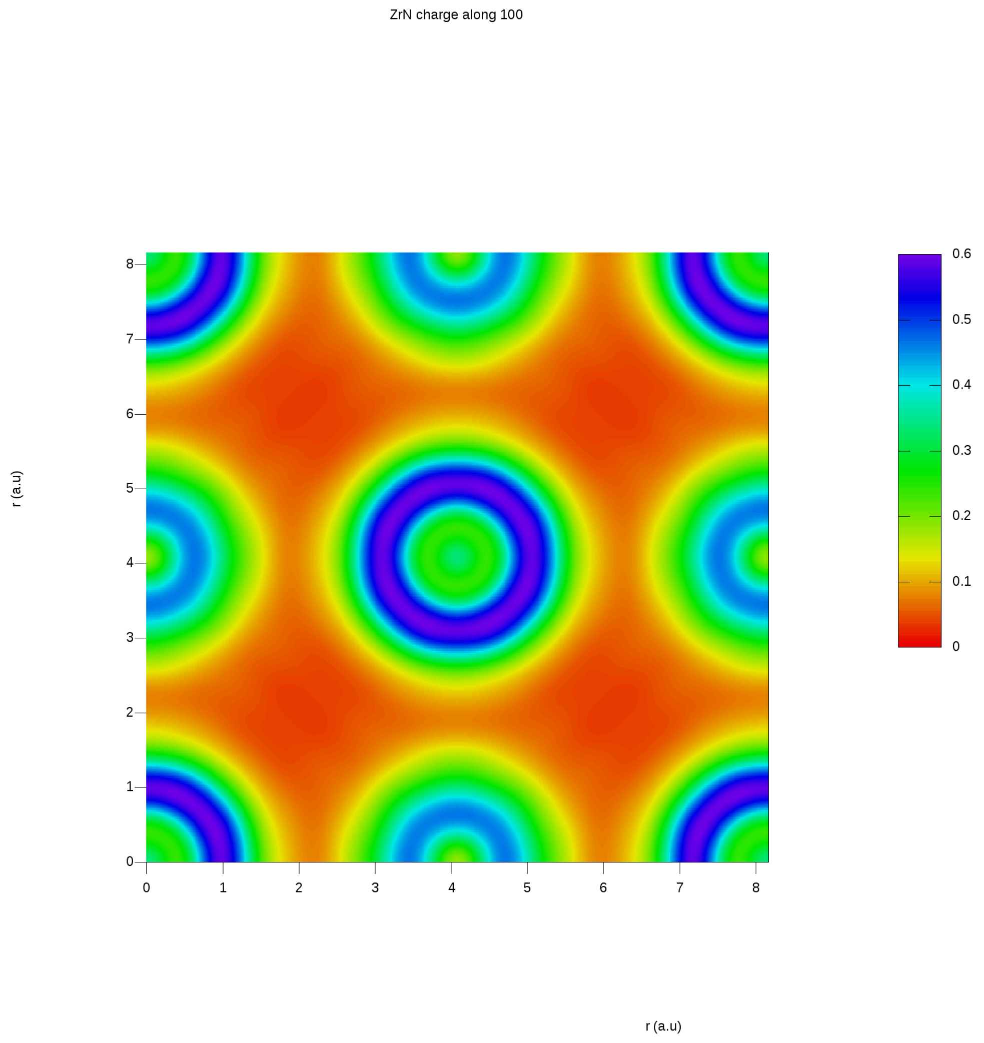
In section four, we calculate the charge density with PP.x for the (110) plane. Here again we need to make sure that e1 and e2 are orthogonal, that the isoval is large enough, that x0(...) is at the origin, and that we set iflag=2 for a 2D plot. We also set output_format = 7 for gnuplot, and plot_num=0 for electron (pseudo-)charge density. The values of nx and ny can be increased for a smoother (more detailed and at the expense of additional 1) storage and 2) computation expense) plot of TiN.charge.png. The original values of nx and ny are 141 and 100 respectively and yield coarse results. Beware however that changing nx and ny may cause the plot TiN.contour.ps to misbehave so if it does, then check that the nx, and ny are not the culprit of any misbehavior before worrying that there is some catastrophic error with the script. Section five plots the output of section four with Gnuplot. For more information on Gnuplot, see some of the specific examples and commentary included in this text in sections 3.8, and 10.5.

Sections six and seven are more or less an identical retelling of sections four and five; however, instead of calculating the charge density on the (110) plane, we calculate on the (100) plane. Likewise, we plot the results of section six with the Gnuplot commands given in section seven. For simplicity, we are only considering a cubic system and therefore we can set alat = celldm(1). The option pm3d is for a 3D Gnuplot file. The script expects that you will set xra (xrange) and yra (yrange) based on the direction that you are simulating; so, for example, in a cubic system, in the (110) direction of a cubic system, set x-range = $[0 : alat * sqrt(2)]$. Alternatively, you could set xra = [0:alat] for (100). Do not forget to end the if statement with fi when editing this portion.

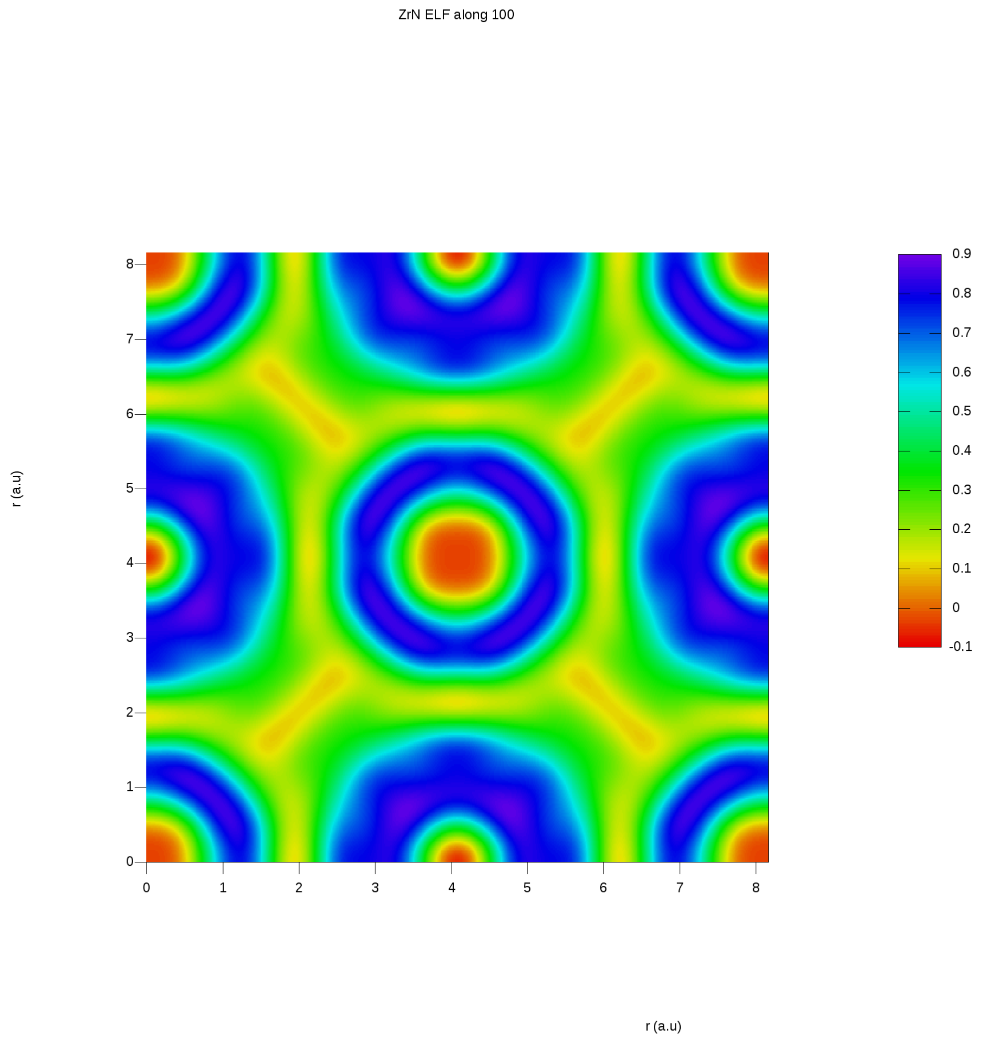
Section eight begins a calculation of the electron localization function with PP.x for the (100) plane. Like before, make sure that e1 and e2 are orthogonal, that the isoval is large enough, that x0(...) is at the origin, and that we set iflag=2 for a 2D plot. We also set output_format = 7 for gnuplot, and plot_num=0 for electron (pseudo-)charge density. The most important part of this section of the script is to set plot_num=8 which is the PP.x input flag for calculating the ELF. As before, you can change the values of nx and ny to give a more or less coarse plot of the ELF. As with other pairs of sections, section nine plots the results of section eight in a familiar way using Gnuplot.

Section ten calculates the electron pseudo charge density using PP.x for the (100) plane. We use similar commands here to many of the other sections previously given. The option spin_component is slightly unique here and is set to 0 which is the option for including the total charge (e.g., contributions of both the spin-up and spin-down charge). Additionally, setting plot_num = 0 is the option to calculate the electron (pseudo-)charge density. Finally, section eleven plots the results of section ten using the familiar Gnuplot commands.

A sample plot of the charge density (this time for ZrN instead of TiN) is given below:



Additionally, a sample plot of the electron localization function (again, for ZrN instead of TiN) is given below:



Please see the following Quantum Espresso calculation script. Remember that the header of the file has been removed and can be added in by copying and pasting (with the application of the appropriate pseudopotential files as well) the header file given in section 6.3.

```
#!/bin/bash

START_TIME=$SECONDS # Begin elapsed time measurement (thanks Tom Anderson from
    StackOverflow)

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:Begin Self-Consistent Calculation:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
```

```
cat > TiN.scf.in << EOF
 &control
                  calculation = 'scf'
                 restart_mode = 'from_scratch',
                       prefix = 'TiN'
                   pseudo_dir = '$PSEUDO_DIR/',
                       outdir = '$TMP_DIR/'
 /
 &system
                        ibrav = 0            ,
                    celldm(1) = 8.00299      ,
                          nat = 2            ,
                         ntyp = 2            ,
                      ecutwfc = 18.0         ,
                  occupations = 'smearing'   ,
                 smearing = 'mv'          ,
                      degauss = 0.02         ,
 /
 &electrons
                     conv_thr = 1.0d-8
                  mixing_beta = 0.7
 /
 CELL_PARAMETERS {alat}
   0.500000000000000    0.500000000000000    0.000000000000000
   0.500000000000000    0.000000000000000    0.500000000000000
   0.000000000000000    0.500000000000000    0.500000000000000
ATOMIC_SPECIES
    Ti    47.86700   Ti.pz-n-nc.UPF
    N     14.00650   N.pz-nc.UPF
 ATOMIC_POSITIONS {crystal}
    Ti    0.000000000000000    0.000000000000000    0.000000000000000
    N     0.500000000000000    0.500000000000000    0.500000000000000
K_POINTS {automatic}
 20 20 20 0 0 0
EOF
$ECHO "  running an scf calculation for TiN...\c"
$PW_COMMAND < TiN.scf.in > TiN.scf.out
check_failure $?
$ECHO " done"

ELAPSED_TIME1=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME1 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-Begin Charge Density Calculation with PP.x:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > TiN.pp_rho.in << EOF
 &inputpp
                       prefix  = 'TiN'
                       outdir = '$TMP_DIR/'
                      filplot = 'TiNcharge'
                     plot_num = 0
 /
 &plot
                        nfile = 1
                    filepp(1) = 'TiNcharge'
                    weight(1) = 1.0
```

```
                        iflag = 2
                output_format = 2
                      fileout = 'TiN.rho.dat'
    e1(1) = 1.0, e1(2) = 1.0, e1(3) = 0.0,
    e2(1) = 0.0, e2(2) = 0.0, e2(3) = 1.0,
    nx=56, ny=40
 /
EOF
$ECHO "  running pp.x to make a 2-d plot of the charge density...\c"
$PP_COMMAND < TiN.pp_rho.in > TiN.pp_rho.out
check_failure $?
$ECHO " done"

ELAPSED_TIME2=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME2 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-Begin Plotrho Commands for electron (pseudo-)charge density-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > TiN.plotrho.in << EOF
TiN.rho.dat
TiN.rho.ps
n
0 0.09 6
EOF

$ECHO "  running plotrho.x to generate rho.ps from the pp.x calculation...\c"
$PLOTRHO_COMMAND < TiN.plotrho.in > TiN.plotrho.out
$ECHO " done"

ELAPSED_TIME3=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME3 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:Begin Charge Density Calculation with PP.x for (110)-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > TiN.pp_rho_110.in << EOF
 &inputpp
                      prefix  = 'TiN'
                       outdir = '$TMP_DIR/'
                      filplot = 'TiNcharge110'
                     plot_num = 0
 /
 &plot
                        nfile = 1
                    filepp(1) = 'TiNcharge110'
                    weight(1) = 1.0
                        iflag = 2
                output_format = 7
                      fileout = 'TiN.rho_110.dat'
    e1(1) = 1.0, e1(2) = 1.0, e1(3) = 0.0,
    e2(1) = 0.0, e2(2) = 0.0, e2(3) = 1.0,
    nx=56, ny=40
 /
EOF

```

```
$ECHO
$ECHO "  running pp.x for a (110) plot of the charge density...\c"
$PP_COMMAND < TiN.pp_rho_110.in > TiN.pp_rho_110.out
check_failure $?
$ECHO " done"

ELAPSED_TIME4=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME4 seconds"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-Begin Plotting with GNUPlot for the (110) direction-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

if [ "$GP_COMMAND" = "" ]; then
    break
else
cat > gnuplot.tmp <<EOF
#!$GP_COMMAND
#
set term png font ",18" enh size 1000,707
set pm3d
set palette model HSV functions gray*0.75, 1, 0.9
set view 0,0
#
alat=8.00299
set xra[0:1.4142136*alat]
set yra [0.:alat]
set size ratio 1./1.4142136
set xtics out nomirror
set ytics axis in offset -4.0,0 nomirror
set label "r (a.u)" at 6.8,-2.2 center
set label "r (a.u)" at -1.7,5.0 rotate by 90 center
unset ztics
unset key
set colorbox
#
set out 'TiN.charge110.png'
set title "TiN charge along 110"
splot 'TiN.rho_110.dat' u 1:2:3 w pm3d
EOF

$ECHO "  generating TiN.charge110.png...\c"
$GP_COMMAND < gnuplot.tmp
$ECHO " done"
#rm gnuplot.tmp

fi

ELAPSED_TIME5=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME5 seconds"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:Begin Charge Density Calculation with PP.x for (100)-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > TiN.pp_rho_100.in << EOF
 &inputpp
                      prefix  = 'TiN'
                       outdir = '$TMP_DIR/'
```

```
                     filplot = 'TiNcharge100'
                    plot_num = 0
 /
 &plot
                       nfile = 1
                   filepp(1) = 'TiNcharge100'
                   weight(1) = 1.0
                       iflag = 2
               output_format = 7
                     fileout = 'TiN.rho_100.dat'
    e1(1) = 1.0, e1(2) = 0.0, e1(3) = 0.0,
    e2(1) = 0.0, e2(2) = 0.0, e2(3) = 1.0,
    nx=56, ny=40
 /
EOF

$ECHO
$ECHO "  running pp.x for a (100) 2D plot of the charge density...\c"
$PP_COMMAND < TiN.pp_rho_100.in > TiN.pp_rho_100.out
check_failure $?
$ECHO " done"

ELAPSED_TIME6=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME6 seconds"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-Begin Plotting with GNUPlot (Assuming That It is Installed)-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

if [ "$GP_COMMAND" = "" ]; then
    break
else
cat > gnuplot.tmp <<EOF
#!$GP_COMMAND
#
set term png font ",18" enh size 1000,1000
set pm3d
set palette model HSV functions gray*0.75, 1, 0.9
set view 0,0
#
alat=8.00299
set xra[0:alat]
set yra [0.:alat]
set size ratio 1./1.
set xtics out nomirror
set ytics axis in offset -4.0,0 nomirror
set label "r (a.u)" at 6.8,-2.2 center
set label "r (a.u)" at -1.7,5.0 rotate by 90 center
unset ztics
unset key
set colorbox
#
set out 'TiN.charge100.png'
set title "TiN charge along 100"
splot 'TiN.rho_100.dat' u 1:2:3 w pm3d
EOF

$ECHO "  generating TiN.charge100.png...\c"
```

```
$GP_COMMAND < gnuplot.tmp
$ECHO " done"
#rm gnuplot.tmp

fi

ELAPSED_TIME7=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME7 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:Begin Electron Localization Function Calculation with PP.x for (100)-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > TiN.pp_ELF_100.in << EOF
 &inputpp
                      prefix  = 'TiN'
                       outdir = '$TMP_DIR/'
                      filplot = 'TiN_ELF_100'
                     plot_num = 8
 /
 &plot
                        nfile = 1
                    filepp(1) = 'TiN_ELF_100'
                    weight(1) = 1.0
                        iflag = 2
                output_format = 7
                      fileout = 'TiN.ELF_100.dat'
    e1(1) = 1.0, e1(2) = 0.0, e1(3) = 0.0,
    e2(1) = 0.0, e2(2) = 0.0, e2(3) = 1.0,
    nx=56, ny=40
 /
EOF

$ECHO
$ECHO "  running pp.x  for a 2D plot of ELF...\c"
$PP_COMMAND < TiN.pp_ELF_100.in > TiN.pp_ELF_100.out
check_failure $?
$ECHO " done"

ELAPSED_TIME8=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME8 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-Begin Plotting with GNUPlot (ELF 100 2D Data Plot):-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

if [ "$GP_COMMAND" = "" ]; then
    break
else
cat > gnuplot.tmp <<EOF
#!$GP_COMMAND
#
set term png font ",18" enh size 1000,1000
set pm3d
set palette model HSV functions gray*0.75, 1, 0.9
set view 0,0
#
```

```
alat=8.00299
set xra[0:alat]
set yra [0.:alat]
set size ratio 1./1.
set xtics out nomirror
set ytics axis in offset -4.0,0 nomirror
set label "r (a.u)" at 6.8,-2.2 center
set label "r (a.u)" at -1.7,5.0 rotate by 90 center
unset ztics
unset key
set colorbox
#
set out 'TiN.ELF100.png'
set title "TiN ELF along 100"
splot 'TiN.ELF_100.dat' u 1:2:3 w pm3d
EOF

$ECHO "  generating TiN.charge100.png...\c"
$GP_COMMAND < gnuplot.tmp
$ECHO " done"
#rm gnuplot.tmp

fi

ELAPSED_TIME9=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME9 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:Begin Electron Pseudo Charge Density Calculation with PP.x for (100)-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > TiN.pp_EpCD_100.in << EOF
 &inputpp
                      prefix  = 'TiN'
                       outdir = '$TMP_DIR/'
                      filplot = 'TiN_EpCD_100'
                     plot_num = 0
             spin_component = 0
 /
 &plot
                        nfile = 1
                    filepp(1) = 'TiN_EpCD_100'
                    weight(1) = 1.0
                        iflag = 2
                output_format = 7
                      fileout = 'TiN.EpCD_100.dat'
    e1(1) = 1.0, e1(2) = 0.0, e1(3) = 0.0,
    e2(1) = 0.0, e2(2) = 0.0, e2(3) = 1.0,
    nx=56, ny=40
 /
EOF

$ECHO
$ECHO "  running pp.x  for a 2D plot of ELF...\c"
$PP_COMMAND < TiN.pp_EpCD_100.in > TiN.pp_EpCD_100.out
check_failure $?
$ECHO " done"

```

```
ELAPSED_TIME10=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME10 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-Begin Plotting with GNUPlot (ELF 100 2D Data Plot):-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

if [ "$GP_COMMAND" = "" ]; then
    break
else
cat > gnuplot.tmp <<EOF
#!$GP_COMMAND
#
set term png font ",18" enh size 1000,1000
set pm3d
set palette model HSV functions gray*0.75, 1, 0.9
set view 0,0
#
alat=8.00299
set xra[0:alat]
set yra [0.:alat]
set size ratio 1./1.
set xtics out nomirror
set ytics axis in offset -4.0,0 nomirror
set label "r (a.u)" at 6.8,-2.2 center
set label "r (a.u)" at -1.7,5.0 rotate by 90 center
unset ztics
unset key
set colorbox
#
set out 'TiN.EpCD100.png'
set title "TiN EpCD along 100"
splot 'TiN.EpCD_100.dat' u 1:2:3 w pm3d
EOF

$ECHO "  generating TiN.charge100.png...\c"
$GP_COMMAND < gnuplot.tmp
$ECHO " done"
#rm gnuplot.tmp

fi

ELAPSED_TIME11=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME11 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-Begin Cleaning Temp Directory-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

$ECHO "  cleaning $TMP_DIR...\c"
rm -rf $TMP_DIR/TiN.*

$ECHO
$ECHO "$EXAMPLE_DIR: done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
```

```
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:End Elapsed Time Measurement-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

ELAPSED_TIME=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME seconds"

echo "Job Completed"
```

6.5 K-resolved projected density of states (KPDOS) and the Fermi surface

As shown in the previous section 6.4, Quantum Espresso (QE) can perform a large variety of calculations. Another useful calculation is that of the k-resolved projected density of states (KPDOS). In broad terms, the projected density of states (PDOS) is the contribution (or the projection) of any one of n orbitals in any one of m atoms to the total density of states (DOS). So, the KPDOS allows us to separate the individual contributions of orbitals to the overall DOS, giving a clear and intuitive picture as to the origin of electronic properties in a system.

In addition to the KPDOS, QE can calculate the Fermi surface. To review some physics, the Fermi surface is the surface in reciprocal space which separates occupied and unoccupied states at zero temperature. In the case that electrons are completely free, the Fermi surface will take the shape of a circle (in the 2D case), or a sphere (in the 3D case). The Fermi surface, like many of the other tools that we discuss in this text, allows the viewer to rapidly gain an intuitive understanding of where some of the electronic properties of a material are coming from. Additionally (and, again somewhat shallowly, the Fermi surface can be very beautiful and can be made into exceptional scientific visualizations with the aid of other codes that will be discussed in section 6.8.

The following script will calculate the KPDOS as well as the Fermi surface of TiN and is divided into nine separate sections which will be described individually (but briefly and as accessibly as reasonable) in the following paragraphs. It is important with these calculations to have a reasonable idea of what the Fermi level is in the material you are calculating. In TiN, I estimate that the Fermi level should be approximately 17.2470 eV. The following script has also been stripped of its header file to be more economical in terms of space; please see section 6.3 for a header file that can (with just a few tweaks of the pseudopotential files) be copied and pasted into the header of the following to make a usable script. Before beginning trying to run the script, make sure that all of your pseudopotentials are readable and that all of the executables you wish to use are called in the header file.

True to the normal form of my QE scripts, I begin here by calculating the SCF in section 1; in this specific instance, I am using kjpaw pseudopotentials. Additionally, part of the input file for this section was generated using the program Cif2Cell which is discussed at length in section 6.7. We create a crystal similar to what was described in 6.4. The option mv Smearing is the 1999 Marzari-Vanderbilt Cold Smearing. The option starting_magnetization gives the starting spin polarization on atom i and is set here, for the sake of the example, to 1 which means that all spins are up. In reality, it is more likely that there will be little to no net magnetic component to TiN. This starting_magnetization is usually more important in SCF calculations than in NSCF calculations. We also set an automatically-defined grid of k-points for the calculation for convenience.

In section two, we begin the band structure calculation. Cell parameters here are not needed and instead just the fractional coordinates of the atoms in the cell are needed. In this case, we manually define 97 k-points along the Δ, and Σ lines. It is important to make sure that ecutwfc, and ecutrho settings are consistent across calculations. Make sure that the value of the degauss setting (which is the Rydberg (Ry) energy value of the gaussian spreading for brillouin-zone integration in metals) is reasonable and realistic.

In section three, we finally begin the k-resolved PDOS calculation. The calculation of the KPDOS will be performed along the lines computed in the band structure calculation in section two (the Δ, and Σ lines here). The option ngauss Is Gaussian Broadening Type, 0 Simple Default. Remember that degauss Is Gaussian Broadening in Ry (not eV). The setting DeltaE Is Energy Grid Step Size in eV and needs to be chosen to be relatively fine but this is (as always) at the expense of increased computational intensity. The heavy lifting here is done with the setting kresolveddos=.true. which informs QE of what type of calculation is being performed specifically.

Section four uses the KPDOS output as its input for plotting all of the results with Gnuplot. For more specific coverage of Gnuplot, please see sections 3.8, and 10.5. This is a somewhat advanced usage of Gnuplot and the script overall uses the multiple plotting engine (called multiplot) to create the graphics of the KPDOS. Make sure to set your specific Fermi energy here as the ef variable. This specific instance of a Gnuplot script will create six plots corresponding to the states that were previously calculated. Depending on the system, there may be more or less states that need to be calculated and therefore the reader may need to modify this part of the script to suit their own needs. Beware however that it can be irritating at this step, in the case that there is a mistake made in one of the edits, the script can terminate and make you begin your calculations again (QE can however restart without calculating everything from scratch). Gnuplot here is set to create .png images because of the png enhanced terminal that was set at the beginning of the script.

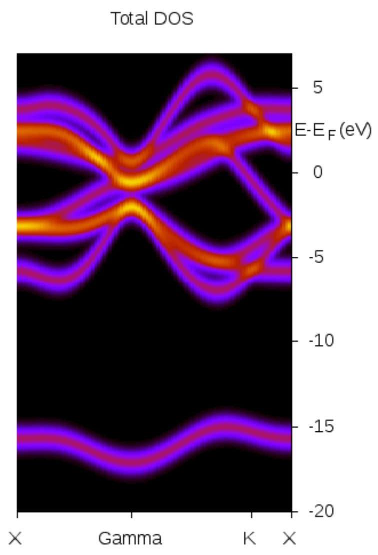
In section five, we begin density of states (DOS) calculations with the calculation of a non-self-consistent field (NSCF). Similarly to section three, the calculation here does not need the specific cell parameter. Like the calculation in section one, this NSCF will be calculated with automatic k-points. We also set occupations to Bloechl's tetrahedral method. The parameter nbnd here is very important and tells QE the number of electronic states that need to be calculated. Choosing a proper value of nbnd is somewhat involved and beyond the scope of this text. However, if you are seeing that one of the .out files returns an error with 'Bad Fermi Energy', then try to increase the value of nbnd to a larger number and that may resolve the error (do this sparingly though and try to not stray from a reasonable number for nbnd).

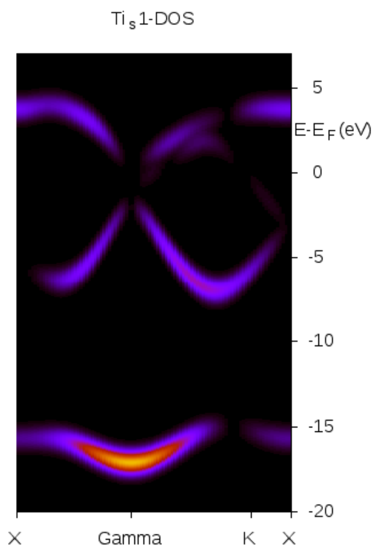
Section five also runs two additional calculations on top of the NSCF: one for the DOS, and one for the PDOS. In both of these calculations, it is important to specify reasonable values for Emin (the minimum energy to calculate), Emax (the maximum energy to calculate), and DeltaE (the step size to calculate). This DeltaE parameter is very important as it specifies the coarseness of the calculation set, reducing this number comes at the expense of a more intense calculation but obviously yields a finer calculation space. It is up to the reader to determine a value of this parameter that is most reasonable for them; however, here I set it as 0.1 which is a common value in these situations.

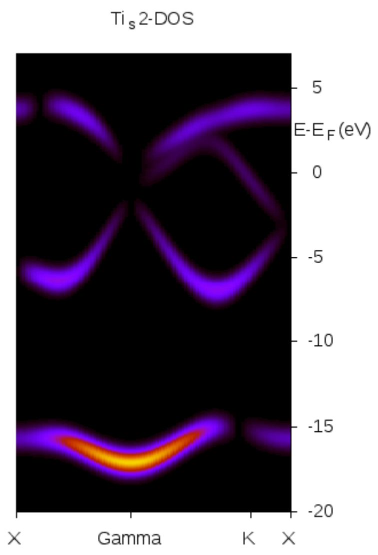
In sections six and seven, we begin the SCF calculation for the spin-polarized (SP) case and then use the output of that calculation to calculate the Fermi surface for the spin-polarized case. Here I choose to use a less coarse grid for the sake of time savings and use an 8x8x8 automatic grid of k-points. The occupations setting is set as smearing which gives gaussian smearing that is suited to metals. This is chosen because TiN is a very good conductor of electricity and has optical properties similar to gold in the visible bandwidth. An NSCF calculation for the SP case is then run in a similar way as before but this time using a much finer grid of k-points. The choice of which calculations can stand to have more or less coarse grids of k-points is important and can be learned but is beyond the scope of this text. If the NSCF calculation halts unexpectedly with an error like "Error condition encountered during test: exit status = 1 Aborting", then try increasing nbnd to a larger number. After the SCF and NSCF for the SP case are calculated, the script will run the Fermi surface calculation using the fs.x FORTRAN code from Quantum Espresso.

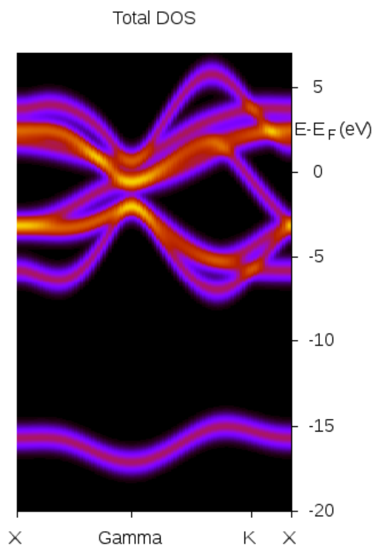
Sections eight and nine do the same sort of thing as sections six and seven, however they calculate the SCF and NSCF of the non-spin-polarized (NSP). The resulting Fermi surfaces will be stored in files with the .bxsf extension and can be visualized rapidly with the program called XCrySDen [5]. These output files will be named in a way similar to the following: TiN_fsXX.bxsf where the XX will be substituted for up or dw (for the up and down cases).

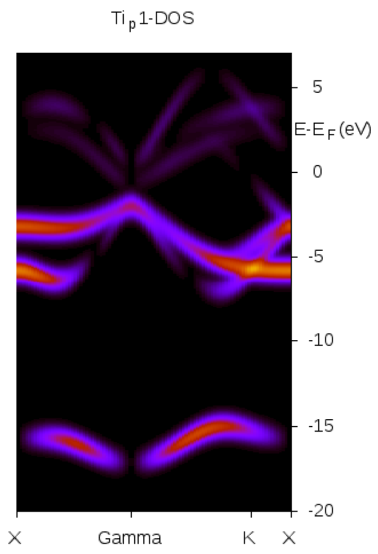
For the TiN up-KPDOS calculations, the results should look something like the following:

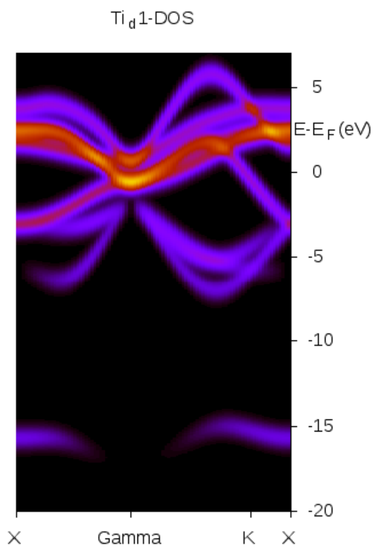

Total DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X

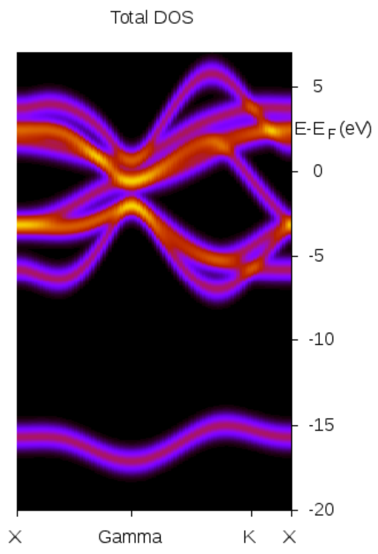

Ti s 1-DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X

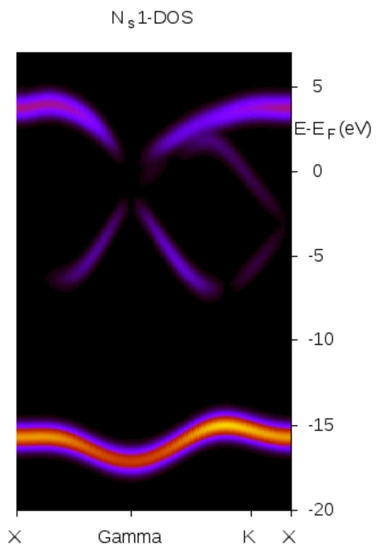

Ti s 2-DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X

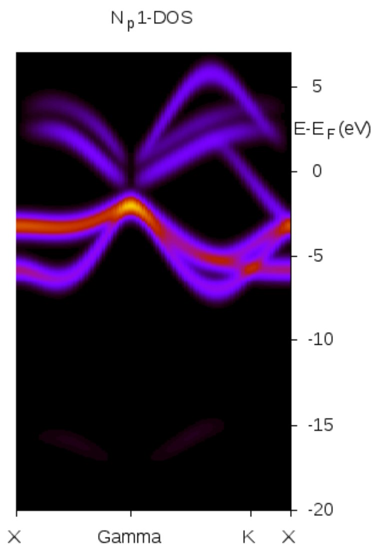

Total DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X


Ti p 1-DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X

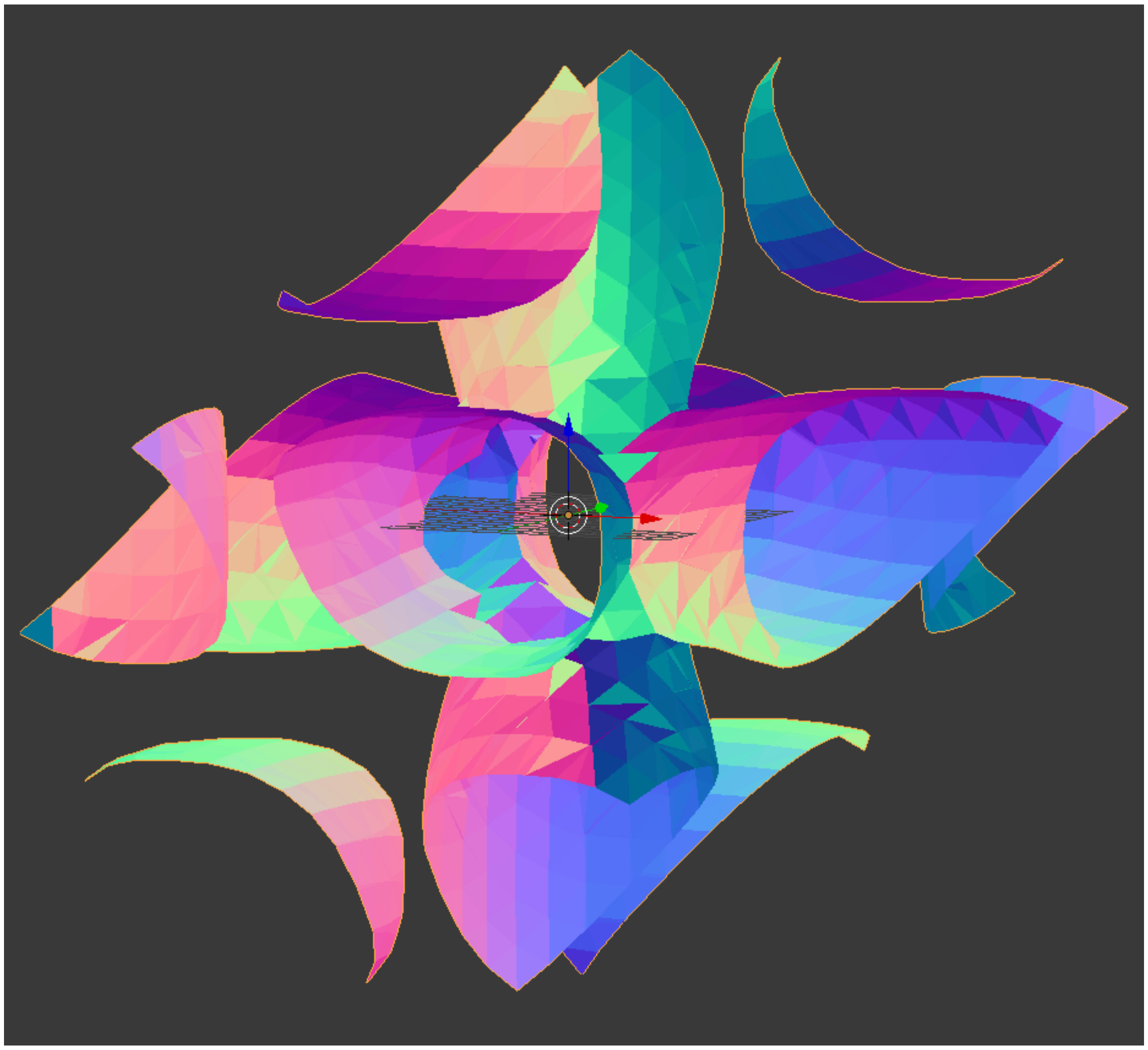

Ti d 1-DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X


Total DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X


N s 1-DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X


N p 1-DOS
5
E-E_F (eV)
0
-5
-10
-15
-20
X
Gamma
K
X

Additionally, for the TiN Fermi surface calculation, the results (after some tinkering with Firmi, a program described in section 6.8, and a 3D graphics program called Blender) should look something like the following:



Please see the following script for the calculation of the KPDOS as well as the Fermi surface. Additionally, see section 6.8 for a quick discussion on how to visualize Fermi surfaces and how to turn them into .stl files for use in 3D graphics programs.

```
#!/bin/sh

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:Begin Self-Consistent Calculation:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
cat > TiN.scf.in << EOF
 &control
     calculation='scf'
     restart_mode='from_scratch',
     prefix='TiN'
     pseudo_dir = '$PSEUDO_DIR/',
     outdir='$TMP_DIR/'
 /
 &SYSTEM
                       ibrav = 0            ,
                   celldm(1) = 8.00299      ,
                         nat = 2            ,
                        ntyp = 2            ,
                     ecutwfc = 40           ,
                     ecutrho = 160          ,
                 occupations = 'smearing'   ,
                smearing = 'mv'          ,
                     degauss = 0.02         ,
                       nspin = 2            ,
   starting_magnetization(1) = 1            ,
   starting_magnetization(2) = 1            ,
           tot_magnetization = -1           ,
 /
 &ELECTRONS
                    conv_thr = 1.0d-8       ,
                 mixing_beta = 0.7          ,
 /
 CELL_PARAMETERS {alat}
   0.500000000000000    0.500000000000000    0.000000000000000
   0.500000000000000    0.000000000000000    0.500000000000000
   0.000000000000000    0.500000000000000    0.500000000000000
 ATOMIC_SPECIES
   Ti    47.86700   Ti.pbe-spn-kjpaw_psl.1.0.0.UPF
   N     14.00650   N.pbe-n-kjpaw_psl.1.0.0.UPF
 ATOMIC_POSITIONS {crystal}
   Ti    0.000000000000000    0.000000000000000    0.000000000000000
   N     0.500000000000000    0.500000000000000    0.500000000000000

# k-space resolution ~0.2/A.
K_POINTS automatic
12 12 12   0 0 0

EOF
$ECHO "running the scf calculation...\c"
$PW_COMMAND < TiN.scf.in > TiN.scf.out
check_failure $?
$ECHO "done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:Begin Band Structure Calculation-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > TiN.band.in << EOF
 &control
     calculation='bands'
     restart_mode='from_scratch',
     prefix='TiN',
     pseudo_dir = '$PSEUDO_DIR/',
     outdir='$TMP_DIR/'
 /
 &SYSTEM
                       ibrav = 2              ,
                   celldm(1) = 8.00299        ,
                         nat = 2              ,
                        ntyp = 2              ,
                     ecutwfc = 40             ,
                     ecutrho = 160            ,
                 occupations = 'smearing'     ,
                smearing = 'mv'           ,
                     degauss = 0.02           ,
                       nspin = 2              ,
   starting_magnetization(1) = 1              ,
   starting_magnetization(2) = 1              ,
           tot_magnetization = -1             ,
 /
 &ELECTRONS
                    conv_thr = 1.0d-8         ,
                 mixing_beta = 0.7            ,
 /
 ATOMIC_SPECIES
   Ti   47.86700  Ti.pbe-spn-kjpaw_psl.1.0.0.UPF
   N    14.00650  N.pbe-n-kjpaw_psl.1.0.0.UPF
 ATOMIC_POSITIONS {crystal}
   Ti   0.000000000000000   0.000000000000000   0.000000000000000
   N    0.500000000000000   0.500000000000000   0.500000000000000
K_POINTS
97
1.000000000  0.000000000  0.000000000  1
0.975000000  0.000000000  0.000000000  2
0.950000000  0.000000000  0.000000000  3
0.925000000  0.000000000  0.000000000  4
0.900000000  0.000000000  0.000000000  5
0.875000000  0.000000000  0.000000000  6
0.850000000  0.000000000  0.000000000  7
0.825000000  0.000000000  0.000000000  8
0.800000000  0.000000000  0.000000000  9
0.775000000  0.000000000  0.000000000  10
0.750000000  0.000000000  0.000000000  11
0.725000000  0.000000000  0.000000000  12
0.700000000  0.000000000  0.000000000  13
0.675000000  0.000000000  0.000000000  14
0.650000000  0.000000000  0.000000000  15
0.625000000  0.000000000  0.000000000  16
0.600000000  0.000000000  0.000000000  17
0.575000000  0.000000000  0.000000000  18
0.550000000  0.000000000  0.000000000  19
0.525000000  0.000000000  0.000000000  20
0.500000000  0.000000000  0.000000000  21
0.475000000  0.000000000  0.000000000  22
0.450000000  0.000000000  0.000000000  23
0.425000000  0.000000000  0.000000000  24
0.400000000  0.000000000  0.000000000  25
0.375000000  0.000000000  0.000000000  26
0.350000000  0.000000000  0.000000000  27
0.325000000  0.000000000  0.000000000  28
0.300000000  0.000000000  0.000000000  29
```

```
0.275000000 0.000000000 0.000000000 30
0.250000000 0.000000000 0.000000000 31
0.225000000 0.000000000 0.000000000 32
0.200000000 0.000000000 0.000000000 33
0.175000000 0.000000000 0.000000000 34
0.150000000 0.000000000 0.000000000 35
0.125000000 0.000000000 0.000000000 36
0.100000000 0.000000000 0.000000000 37
0.075000000 0.000000000 0.000000000 38
0.050000000 0.000000000 0.000000000 39
0.025000000 0.000000000 0.000000000 40
0.000000000 0.000000000 0.000000000 41
0.017857142 0.017857142 0.000000000 42
0.035714285 0.035714285 0.000000000 43
0.053571428 0.053571428 0.000000000 44
0.071428571 0.071428571 0.000000000 45
0.089285714 0.089285714 0.000000000 46
0.107142857 0.107142857 0.000000000 47
0.125000000 0.125000000 0.000000000 48
0.142857142 0.142857142 0.000000000 49
0.160714285 0.160714285 0.000000000 50
0.178571428 0.178571428 0.000000000 51
0.196428571 0.196428571 0.000000000 52
0.214285714 0.214285714 0.000000000 53
0.232142857 0.232142857 0.000000000 54
0.250000000 0.250000000 0.000000000 55
0.267857142 0.267857142 0.000000000 56
0.285714285 0.285714285 0.000000000 57
0.303571428 0.303571428 0.000000000 58
0.321428571 0.321428571 0.000000000 59
0.339285714 0.339285714 0.000000000 60
0.357142857 0.357142857 0.000000000 61
0.375000000 0.375000000 0.000000000 62
0.392857142 0.392857142 0.000000000 63
0.410714285 0.410714285 0.000000000 64
0.428571428 0.428571428 0.000000000 65
0.446428571 0.446428571 0.000000000 66
0.464285714 0.464285714 0.000000000 67
0.482142857 0.482142857 0.000000000 68
0.500000000 0.500000000 0.000000000 69
0.517857142 0.517857142 0.000000000 70
0.535714285 0.535714285 0.000000000 71
0.553571428 0.553571428 0.000000000 72
0.571428571 0.571428571 0.000000000 73
0.589285714 0.589285714 0.000000000 74
0.607142857 0.607142857 0.000000000 75
0.625000000 0.625000000 0.000000000 76
0.642857142 0.642857142 0.000000000 77
0.660714285 0.660714285 0.000000000 78
0.678571428 0.678571428 0.000000000 79
0.696428571 0.696428571 0.000000000 80
0.714285714 0.714285714 0.000000000 81
0.732142857 0.732142857 0.000000000 82
0.750000000 0.750000000 0.000000000 83
0.767857142 0.767857142 0.000000000 84
0.785714285 0.785714285 0.000000000 85
0.803571428 0.803571428 0.000000000 86
0.821428571 0.821428571 0.000000000 87
```

```
0.839285714 0.839285714 0.000000000 88
0.857142857 0.857142857 0.000000000 89
0.875000000 0.875000000 0.000000000 90
0.892857142 0.892857142 0.000000000 91
0.910714285 0.910714285 0.000000000 92
0.928571428 0.928571428 0.000000000 93
0.946428571 0.946428571 0.000000000 94
0.964285714 0.964285714 0.000000000 95
0.982142857 0.982142857 0.000000000 96
1.000000000 1.000000000 0.000000000 97

EOF
$ECHO "  running the band-structure calculation for TiN...\c"
$PW_COMMAND < TiN.band.in > TiN.band.out
check_failure $?
$ECHO " done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-Begin K-resolved PDOS calculation along lines computed above:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > TiN.kpdos.in << EOF
 &projwfc
    outdir='$TMP_DIR/'
    prefix='TiN'
    ngauss=0, degauss=0.036748
    DeltaE=0.01
    kresolveddos=.true.
    filpdos='TiN.k'
 /
EOF
$ECHO "  running k-resolved PDOS calculation for TiN...\c"
$PROJWFC_COMMAND < TiN.kpdos.in > TiN.kpdos.out
check_failure $?
$ECHO " done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-Begin Plotting with GNUPlot (Assuming That It is Installed)-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

if [ "$GP_COMMAND" = "" ]; then
    break
else
cat > gnuplot.tmp <<EOF
#!$GP_COMMAND
#
set term png enh size 1000,500
set pm3d
set view 0,0
#
f(z)=z**(0.7)  # tune image contrast
ef=17.2470
#
unset xtics
set xtics out nomirror ("X" 1,"Gamma" 41,"K" 83, "X" 97)
set xra[1:97]
set label 1 "E-E_F(eV)" at 98,2.5
set ytics out nomirror
set yra [-20:7]
unset ztics
unset key
unset colorbox

#
set out 'kpdos_up1.png'
set origin 0,0
set size 1,1
set multiplot
dx=.1 ; dy=.30   # reduce margins
set title offset 0,-7
set size 1./3+1.4*dx,1.+2*dy
set origin 0./3-dx,0-dy
set title "Total DOS"
splot 'TiN.k.pdos_tot'           u 1:(\$2-ef):(f(\$3)) w pm3d
set origin 1./3-dx,0-dy
set title "Ti_s1-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#1(s)' u 1:(\$2-ef):(f(\$3)) w pm3d
set origin 2./3-dx,0-dy
set title "Ti_s2-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#2(s)' u 1:(\$2-ef):(f(\$3)) w pm3d
unset multiplot

#
set out 'kpdos_up2.png'
set origin 0,0
set size 1,1
set multiplot
dx=.1 ; dy=.30   # reduce margins
set title offset 0,-7
set size 1./3+1.4*dx,1.+2*dy
set origin 0./3-dx,0-dy
set title " Total DOS"
splot 'TiN.k.pdos_tot'           u 1:(\$2-ef):(f(\$3)) w pm3d
set origin 1./3-dx,0-dy
set title "Ti_p1-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#3(p)' u 1:(\$2-ef):(f(\$3)) w pm3d
set origin 2./3-dx,0-dy
set title "Ti_d1-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#4(d)' u 1:(\$2-ef):(f(\$3)) w pm3d
unset multiplot

#
set out 'kpdos_up3.png'
set origin 0,0
set size 1,1
set multiplot
dx=.1 ; dy=.30   # reduce margins
set title offset 0,-7
set size 1./3+1.4*dx,1.+2*dy
set origin 0./3-dx,0-dy
set title "Total DOS"
splot 'TiN.k.pdos_tot'           u 1:(\$2-ef):(f(\$3)) w pm3d
set origin 1./3-dx,0-dy
set title "N_s1-DOS"
splot 'TiN.k.pdos_atm#2(N)_wfc#1(s)'  u 1:(\$2-ef):(f(\$3)) w pm3d
set origin 2./3-dx,0-dy
set title "N_p1-DOS"
splot 'TiN.k.pdos_atm#2(N)_wfc#2(p)'  u 1:(\$2-ef):(f(\$3)) w pm3d
unset multiplot

#
set out 'kpdos_dw1.png'
set origin 0,0
set size 1,1
set multiplot
dx=.1 ; dy=.30   # reduce margins
set title offset 0,-7
set size 1./3+1.4*dx,1.+2*dy
set origin 0./3-dx,0-dy
set title "Total DOS"
splot 'TiN.k.pdos_tot'           u 1:(\$2-ef):(f(\$4)) w pm3d
set origin 1./3-dx,0-dy
set title "Ti_s1-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#1(s)' u 1:(\$2-ef):(f(\$4)) w pm3d
set origin 2./3-dx,0-dy
set title "Ti_s2-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#2(s)' u 1:(\$2-ef):(f(\$4)) w pm3d
unset multiplot

#
set out 'kpdos_dw2.png'
set origin 0,0
set size 1,1
set multiplot
dx=.1 ; dy=.30   # reduce margins
set title offset 0,-7
set size 1./3+1.4*dx,1.+2*dy
set origin 0./3-dx,0-dy
set title " Total DOS"
splot 'TiN.k.pdos_tot'           u 1:(\$2-ef):(f(\$4)) w pm3d
set origin 1./3-dx,0-dy
set title "Ti_p1-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#3(p)' u 1:(\$2-ef):(f(\$4)) w pm3d
set origin 2./3-dx,0-dy
set title "Ti_d1-DOS"
splot 'TiN.k.pdos_atm#1(Ti)_wfc#4(d)' u 1:(\$2-ef):(f(\$4)) w pm3d
unset multiplot

#
set out 'kpdos_dw3.png'
set origin 0,0
set size 1,1
set multiplot
dx=.1 ; dy=.30   # reduce margins
set title offset 0,-7
set size 1./3+1.4*dx,1.+2*dy
set origin 0./3-dx,0-dy
set title "Total DOS"
splot 'TiN.k.pdos_tot'           u 1:(\$2-ef):(f(\$4)) w pm3d
set origin 1./3-dx,0-dy
set title "N_s1-DOS"
splot 'TiN.k.pdos_atm#2(N)_wfc#1(s)'  u 1:(\$2-ef):(f(\$4)) w pm3d
set origin 2./3-dx,0-dy
set title "N_p1-DOS"
```

```
splot 'TiN.k.pdos_atm#2(N)_wfc#2(p)'  u 1:(\$2-ef):(f(\$4)) w pm3d
unset multiplot

EOF
$ECHO
$ECHO "  plotting k-resolved DOS ...\c"
$GP_COMMAND < gnuplot.tmp
$ECHO " done"
rm gnuplot.tmp
fi

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-Begin Density of States Calculation-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > TiN.dos.in << EOF
 &control
     calculation='nscf'
     prefix='TiN',
     pseudo_dir = '$PSEUDO_DIR/',
     outdir='$TMP_DIR/'
 /
 &SYSTEM
                        ibrav = 2               ,
                    celldm(1) = 8.00299         ,
                          nat = 2               ,
                         ntyp = 2               ,
                      ecutwfc = 40              ,
                      ecutrho = 160             ,
                  occupations = 'tetrahedra'    ,
                      degauss = 0.02            ,
                        nspin = 2               ,
   starting_magnetization(1) = 1               ,
   starting_magnetization(2) = 1               ,
           tot_magnetization = -1              ,
                         nbnd = 10              ,
 /
 &electrons
                     conv_thr = 1.0d-8          ,
                  mixing_beta = 0.7             ,
 /
 ATOMIC_SPECIES
   Ti   47.86700   Ti.pbe-spn-kjpaw_psl.1.0.0.UPF
   N    14.00650   N.pbe-n-kjpaw_psl.1.0.0.UPF
 ATOMIC_POSITIONS {crystal}
   Ti   0.000000000000000   0.000000000000000   0.000000000000000
   N    0.500000000000000   0.500000000000000   0.500000000000000
K_POINTS {automatic}
 12 12 12 0 0 0
EOF

cat > TiN.dos2.in << EOF
 &dos
     outdir='$TMP_DIR/'
     prefix='TiN'
     fildos='TiN.dos',
     Emin=5.0, Emax=25.0, DeltaE=0.1
 /
```

```
EOF

$ECHO "  running DOS calculation for TiN...\c"
$PW_COMMAND < TiN.dos.in > TiN.dos.out
check_failure $?
$DOS_COMMAND < TiN.dos2.in > TiN.dos2.out
check_failure $?
$ECHO " done"

cat > TiN.pdos.in << EOF
 &projwfc
    outdir='$TMP_DIR/'
    prefix='TiN'
    Emin=5.0, Emax=25.0, DeltaE=0.1
    ngauss=1, degauss=0.02
 /
EOF
$ECHO "  running PDOS calculation for TiN...\c"
$PROJWFC_COMMAND < TiN.pdos.in > TiN.pdos.out
check_failure $?
$ECHO " done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:Update User for Fermi Surface Section:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

$ECHO
$ECHO "  Beginning Calculations for the Fermi Surface plot, "
$ECHO "  Spin-Polarized case..."

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:Begin Self-Consistent Calculation for the Spin-Polarized (SP) Case-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > TiN.scf_SP.in << EOF
 &control
    calculation='scf'
    restart_mode='from_scratch',
    prefix='TiN',
    pseudo_dir = '$PSEUDO_DIR/',
    outdir='$TMP_DIR/'
 /
 &SYSTEM
                       ibrav = 2                   ,
                   celldm(1) = 8.00299             ,
                         nat = 2                   ,
                        ntyp = 2                   ,
                     ecutwfc = 40                  ,
                     ecutrho = 160                 ,
                 occupations = 'smearing'          ,
                    smearing = 'methfessel-paxton',
                     degauss = 0.02                ,
                       nspin = 2                   ,
    starting_magnetization(1) = 1                  ,
    starting_magnetization(2) = 1                  ,
            tot_magnetization = -1                 ,
 /
 &electrons
```

```
                     conv_thr = 1.0d-8                ,
                  mixing_beta = 0.7                   ,
 /
 ATOMIC_SPECIES
   Ti    47.86700   Ti.pbe-spn-kjpaw_psl.1.0.0.UPF
   N     14.00650   N.pbe-n-kjpaw_psl.1.0.0.UPF
 ATOMIC_POSITIONS {crystal}
   Ti    0.000000000000000    0.000000000000000    0.000000000000000
   N     0.500000000000000    0.500000000000000    0.500000000000000
K_POINTS {automatic}
 8 8 8 0 0 0
EOF
$ECHO "  running the scf calculation for the spin-polarized case ... \c"
$PW_COMMAND < TiN.scf_SP.in > TiN.scf0.SP.out
check_failure $?
$ECHO " done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-Begin Fermi Surface Calculation for the Spin-Polarized (SP) Case:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > TiN.fs_SP.in << EOF
 &control
     calculation='nscf'
     prefix='TiN',
     pseudo_dir = '$PSEUDO_DIR/',
     outdir='$TMP_DIR/'
 /
 &SYSTEM
                        ibrav = 2             ,
                    celldm(1) = 8.00299       ,
                          nat = 2             ,
                         ntyp = 2             ,
                       ecutwfc = 40            ,
                       ecutrho = 160           ,
                  occupations = 'tetrahedra' ,
                         nbnd = 10            ,
                        nspin = 2             ,
   starting_magnetization(1) = 1             ,
   starting_magnetization(2) = 1             ,
           tot_magnetization = -1            ,
 /
 &electrons
                     conv_thr = 1.0d-8        ,
                  mixing_beta = 0.7           ,
 /
 ATOMIC_SPECIES
   Ti    47.86700   Ti.pbe-spn-kjpaw_psl.1.0.0.UPF
   N     14.00650   N.pbe-n-kjpaw_psl.1.0.0.UPF
 ATOMIC_POSITIONS {crystal}
   Ti    0.000000000000000    0.000000000000000    0.000000000000000
   N     0.500000000000000    0.500000000000000    0.500000000000000
K_POINTS {automatic}
16 16 16 0 0 0
EOF

$ECHO "  running the Fermi Surface calculation ... \c"
$PW_COMMAND   < TiN.fs_SP.in > TiN.fs_SP.out
```

```
check_failure $?
$ECHO " done"

cat > FS.in <<EOF
&fermi
    outdir='$TMP_DIR/'
    prefix='TiN'
/
EOF
$FS_COMMAND < FS.in > FS.out
check_failure $?

$ECHO
$ECHO "  Use 'xcrysden --bxsf results/TiN_fsXX.bxsf', XX=up,dw to plot the Fermi
    Surface\c"
$ECHO " done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:Update User for Fermi Surface Section:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

$ECHO
$ECHO "  Beginning Calculations for the Fermi Surface plot, "
$ECHO "  Non-Spin-Polarized case..."

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-Begin Self-Consistent Calculation for the Non-Spin-Polarized (NSP) Case-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > TiN.scf_NSP.in << EOF
 &control
    calculation='scf'
    restart_mode='from_scratch',
    prefix='TiN',
    pseudo_dir = '$PSEUDO_DIR/',
    outdir='$TMP_DIR/'
 /
 &SYSTEM
                       ibrav = 2                    ,
                   celldm(1) = 8.00299              ,
                         nat = 2                    ,
                        ntyp = 2                    ,
                     ecutwfc = 40                   ,
                     ecutrho = 160                  ,
                 occupations = 'smearing'           ,
                    smearing = 'methfessel-paxton',
                     degauss = 0.02                 ,
                        nbnd = 10                   ,
                       nspin = 2                    ,
    starting_magnetization(1) = 1                   ,
    starting_magnetization(2) = 1                   ,
           tot_magnetization = -1                   ,
 /
 &electrons
                    conv_thr = 1.0d-8               ,
                 mixing_beta = 0.7                  ,
 /
 ATOMIC_SPECIES
```

```
   Ti    47.86700   Ti.pbe-spn-kjpaw_psl.1.0.0.UPF
   N     14.00650   N.pbe-n-kjpaw_psl.1.0.0.UPF
 ATOMIC_POSITIONS {crystal}
   Ti    0.000000000000000    0.000000000000000    0.000000000000000
   N     0.500000000000000    0.500000000000000    0.500000000000000
K_POINTS {automatic}
 8 8 8 0 0 0
EOF
$ECHO "  running the scf calculation  non spin-polarized case ... \c"
$PW_COMMAND < TiN.scf_NSP.in > TiN.scf0.NSP.out
check_failure $?
$ECHO " done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-Begin Fermi Surface Calculation for the Spin-Polarized (NSP) Case-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > TiN.fs_NSP.in << EOF
 &control
    calculation='nscf'
    prefix='TiN',
    pseudo_dir = '$PSEUDO_DIR/',
    outdir='$TMP_DIR/'
 /
 &SYSTEM
                       ibrav = 2              ,
                   celldm(1) = 8.00299        ,
                         nat = 2              ,
                        ntyp = 2              ,
                     ecutwfc = 40             ,
                     ecutrho = 160            ,
                 occupations = 'tetrahedra'   ,
                        nbnd = 10             ,
 /
 &electrons
                    conv_thr = 1.0d-8         ,
                 mixing_beta = 0.7            ,
 /
 ATOMIC_SPECIES
   Ti    47.86700   Ti.pbe-spn-kjpaw_psl.1.0.0.UPF
   N     14.00650   N.pbe-n-kjpaw_psl.1.0.0.UPF
 ATOMIC_POSITIONS {crystal}
   Ti    0.000000000000000    0.000000000000000    0.000000000000000
   N     0.500000000000000    0.500000000000000    0.500000000000000
K_POINTS automatic
16 16 16 0 0 0
EOF

$ECHO "  running the Fermi Surface calculation ... \c"
$PW_COMMAND   < TiN.fs_NSP.in > TiN.fs_NSP.out
check_failure $?
$ECHO " done"

cat > FS.in <<EOF
&fermi
    outdir='$TMP_DIR/'
    prefix='TiN'
/
```

```
EOF
$FS_COMMAND < FS.in > FS.out
check_failure $?

$ECHO
$ECHO "  Use 'xcrysden --bxsf results/TiN_fs.bxsf' to plot the Fermi Surface\c"
$ECHO " done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-Begin Cleaning Temp Directory-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

$ECHO "  cleaning $TMP_DIR...\c"
rm -rf $TMP_DIR/TiN.*

$ECHO
$ECHO "$EXAMPLE_DIR: done"
```

6.6 The complex dielectric function with epsilon.x

My interest in calculating complex dielectric functions stems from my own Ph.D. research work on engineering the optical properties of materials and metasurfaces using chemistry and structuring to enhance the tunability of permittivity. Experimentally, there are two common ways of determining the permittivity of a material: one is to use ellipsometry and subsequent mathematical fitting; another means is to measure reflection and transmission of light through a film and, with precise knowledge of the thickness of the film, you use something like the reverse Newton's method to iteratively determine an n, and k that will suit the R, and T values that you give your code for every data point in your measurement bandwidth based on some model like the Drude model.

Calculating the complex dielectric function of an arbitrary structure based on DFT is difficult and needs to be paired generally with a very rigorous GW calculation to get reasonable results. However, as a starting point for the user, one can attempt this calculation (as it is described here I do not include the GW calculation but the user can do that on their own time with a suitable code like BerkeleyGW on a supercomputer of sufficient power) just based on the self consistent and non self consistent fields (denoted as SCF and NSCF respectively herein).

Quantum espresso has an executable for calculating the complex dielectric function that is called epsilon.x. The minutia of how exactly this is calculation done and how epsilon.x works is supplied within the epsilon.x documentation with distributions of Quantum Espresso and is best left to the authors to describe. As a word of warning, be sure to verify that the results of calculations match reasonably with reality. Without the GW calculation corrections here, the results will not be representative of a realistic material in most cases. That being said, I believe the utility of supplying this script is to add some documentation to the implementation of epsilon.x and the potential sources of errors that can occur with its use. This calculation is light-weight in terms of computational resources and obviously anemic when it comes to the density of the grid over which the calculation is performed as well as having not very stringent requirements for the convergence threshold. The user will have to tune these parameters to suit their own individual needs.

The following script is separated into five general sections. For the sake of brevity, I have omitted the header of the file which includes operations like locating the environment variables and other things. The exact header file that I used for this script is included above in section 6.3 and can be copied and pasted into the header of the following script.

Before working lots on Quantum Espresso (or at leas concurrently with starting), I highly recommend reading selections from the documentation to make sure that you know exactly what every step within your calculation is doing.

Section one of this script runs the SCF calculation using a norm-conserving pseudopotential (because, at least at the time of this writing, epsilon.x will only work on SCF and NSCF calculations run with norm-conserving pseudopotentials. In the case of epsilon.x, we need to set the number of processors as constant so we include the command wf_collect=.TRUE. Additionally, make sure that the ecutwfc and ecutrho (energy terms) values are consistent between the SCF and NSCF calculations. At the time of this writing, epsilon.x does not support an irreducible unit cell and, in this case, I am using a single atom unit cell with one Au atom centered at the origin.

The unit cell is given ibrav = 0 and celldm(1) = 5.44951 which means that we can define our unit cell and atomic positions within the script: all of that is handled in terms of the lattice constant. 60 points in reciprocal space are given and manually-defined (with the aid of a computer program, of course) for the calculation in the order of $\Gamma \to X \to W \to K \to L \to \Gamma$. Finally, the script is ended and the cat command is used to write the script into a file. We then redirect the script into an executable that we defined in the header of the file and stored in a variable and redirect the output of that variable into another file called Au.scf.out and check to see if there was a failure.

Section two is very similar to section one but instead we are calculating the NSCF instead of the SCF. Like in the SCF instructions, we give pertinent parameters for the NSCF at the beginning in the &control, the &system, and the &electrons sections. We have also made sure that the ecutwfc and ecutrho are consistent with the SCF calculation. Other parameters can be perused at the leisure of the reader within the documentation of Quantum Espresso. Finally, we define the same crystal as in the SCF calculation with the same pseudopotential and give the same k-points for calculation steps as in the SCF calculation. We also submit the calculation to the computer in the same way as before in the SCF calculation but this time we redirect the executable's output into a new file named Au.nscf.out.

In section three, we're finally ready to run the epsilon.x calculation. Several pertinent terms within the calculation are as follows: intrasmear is the broadening parameter for the intraband i.e., Drude-like term in eV; intersmear is the broadening parameter (in eV) for the interband contribution and its default is 0.136d0 (eV); nw is the number of points in the frequency mesh. This calculation will be performed on the interval [-wmax,wmax], wmax units of eV; shift is number of eV for rigid shift of the imaginary part of the wavefunction. For Au, I have estimated that the threshold for interband transitions is approximately 2.5 eV.

The epsilon.x program has input and output following the same standard as other Quantum Espresso executables and we will run it, as well as write its output in a more or less identical manner to what I have described above. In the event that epsilon.x returns an error regarding division by zero, then there is likely an issue with the intersmear parameter being too small which leads to strange infinities.

In the fourth section, I include a data file that I call Au.dat which is automatically written to the directory and includes the wavelength in nm, n, k, energy (eV), and the real, and imaginary parts of the complex dielectric function of Au as given in P. B. Johnson and R. W. Christy. Optical constants of the noble metals, Phys. Rev. B 6, 4370-4379 (1972). This is included for comparison purposes of the calculated values with real, experimentally-measured values of the complex dielectric function of Au. Having a comparison data set like this is very important because it helps to rationalize results and have a good comparison to an actual, physical result so that you can more rapidly judge the relative accuracy of these calculations.

Finally, in the fifth section, we create several plots using GNUplot and write them to various png files for comparison with the Johnson and Christy values of the measured Au permittivity. This is done by setting up a script that will be interpreted by GNUplot and then passing that script to GNUplot so it can create all of its nice graphics for you. This is done in a similar way to other scripts I list in this text. Again, for reference, see section 3.8 for more regarding the use of GNUplot.

Additionally, at every step along this calculation, I include a timer program that prints the time that each step in the calculation has taken to reach its completion to the terminal. I think that this little bit of functionality is very nice to have on hand for benchmarking purposes but also just for the purpose of monitoring the calculation and what stage it is currently working on. Make sure that you are running this script in the BASH environment or the timer functionality will not work properly.

```
    START_TIME=$SECONDS # Begin elapsed time measurement (thanks Tom Anderson from
    StackOverflow)

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:Begin Calculation Directions-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:Begin Self-Consistent Calculation:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:.in File partially Generated with Cif2Cell for the Au B1 System-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-Gaussian Smearing is Suited for Metals:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-mv Smearing is the 1999 Marzari-Vanderbilt Cold Smearing:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:for the SCF and the NSCF calculations:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > Au.scf.in << EOF
 &control
    calculation='scf'
    restart_mode='from_scratch',
    prefix='Au'
    pseudo_dir = '$PSEUDO_DIR/',
    outdir='$TMP_DIR/'
    wf_collect=.TRUE,
 /
 &SYSTEM
                       ibrav = 0           ,
                   celldm(1) = 5.44951     ,
                         nat = 1           ,
                        ntyp = 1           ,
                     ecutwfc = 65          ,
                     ecutrho = 260         ,
                 occupations = 'smearing'  ,
                smearing = 'mv'         ,
                     degauss = 0.02        ,
 /
 &ELECTRONS
                    conv_thr = 1.0d-8      ,
                 mixing_beta = 0.7         ,
 /
CELL_PARAMETERS {alat}
  0.577350269189626   0.000000000000000   0.816496580927725
 -0.288675134594812  -0.500000000000000   0.816496580927725
 -0.288675134594812   0.500000000000000   0.816496580927725
```

```
ATOMIC_SPECIES
  Au   196.96600   Au.rel-pbesol-n-nc.UPF
ATOMIC_POSITIONS {crystal}
Au    0.000000000000000    0.000000000000000    0.000000000000000

 K_POINTS crystal
          60
         0.0000000000     0.0000000000     0.0000000000     1.0
         0.0000000000     0.0277777778     0.0277777778     1.0
         0.0000000000     0.0555555556     0.0555555556     1.0
         0.0000000000     0.0833333333     0.0833333333     1.0
         0.0000000000     0.1111111111     0.1111111111     1.0
         0.0000000000     0.1388888889     0.1388888889     1.0
         0.0000000000     0.1666666667     0.1666666667     1.0
         0.0000000000     0.1944444444     0.1944444444     1.0
         0.0000000000     0.2222222222     0.2222222222     1.0
         0.0000000000     0.2500000000     0.2500000000     1.0
         0.0000000000     0.2777777778     0.2777777778     1.0
         0.0000000000     0.3055555556     0.3055555556     1.0
         0.0000000000     0.3333333333     0.3333333333     1.0
         0.0000000000     0.3611111111     0.3611111111     1.0
         0.0000000000     0.3888888889     0.3888888889     1.0
         0.0000000000     0.4166666667     0.4166666667     1.0
         0.0000000000     0.4444444444     0.4444444444     1.0
         0.0000000000     0.4722222222     0.4722222222     1.0
         0.0000000000     0.5000000000     0.5000000000     1.0
         0.0277777778     0.5277777778     0.5000000000     1.0
         0.0555555556     0.5555555556     0.5000000000     1.0
         0.0833333333     0.5833333333     0.5000000000     1.0
         0.1111111111     0.6111111111     0.5000000000     1.0
         0.1388888889     0.6388888889     0.5000000000     1.0
         0.1666666667     0.6666666667     0.5000000000     1.0
         0.1944444444     0.6944444444     0.5000000000     1.0
         0.2222222222     0.7222222222     0.5000000000     1.0
         0.2500000000     0.7500000000     0.5000000000     1.0
         0.2708333333     0.7500000000     0.4791666667     1.0
         0.2916666667     0.7500000000     0.4583333333     1.0
         0.3125000000     0.7500000000     0.4375000000     1.0
         0.3333333333     0.7500000000     0.4166666667     1.0
         0.3541666667     0.7500000000     0.3958333333     1.0
         0.3750000000     0.7500000000     0.3750000000     1.0
         0.3863636364     0.7272727273     0.3863636364     1.0
         0.3977272727     0.7045454545     0.3977272727     1.0
         0.4090909091     0.6818181818     0.4090909091     1.0
         0.4204545455     0.6590909091     0.4204545455     1.0
         0.4318181818     0.6363636364     0.4318181818     1.0
         0.4431818182     0.6136363636     0.4431818182     1.0
         0.4545454545     0.5909090909     0.4545454545     1.0
         0.4659090909     0.5681818182     0.4659090909     1.0
         0.4772727273     0.5454545455     0.4772727273     1.0
         0.4886363636     0.5227272727     0.4886363636     1.0
         0.5000000000     0.5000000000     0.5000000000     1.0
         0.4666666667     0.4666666667     0.4666666667     1.0
         0.4333333333     0.4333333333     0.4333333333     1.0
         0.4000000000     0.4000000000     0.4000000000     1.0
         0.3666666667     0.3666666667     0.3666666667     1.0
         0.3333333333     0.3333333333     0.3333333333     1.0
         0.3000000000     0.3000000000     0.3000000000     1.0
```

```
        0.2666666667    0.2666666667    0.2666666667    1.0
        0.2333333333    0.2333333333    0.2333333333    1.0
        0.2000000000    0.2000000000    0.2000000000    1.0
        0.1666666667    0.1666666667    0.1666666667    1.0
        0.1333333333    0.1333333333    0.1333333333    1.0
        0.1000000000    0.1000000000    0.1000000000    1.0
        0.0666666667    0.0666666667    0.0666666667    1.0
        0.0333333333    0.0333333333    0.0333333333    1.0
        0.0000000000    0.0000000000    0.0000000000    1.0

EOF
$ECHO "running the scf calculation...\c"
$PW_COMMAND < Au.scf.in > Au.scf.out
check_failure $?
$ECHO "done"

ELAPSED_TIME1=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME1 seconds"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:Begin Non-Self-Consistent Calculation:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:Use Norm-Conserving Pseudopotentials for epsilon.x to work-:-:-:-:-
#-:-:-:-:-:-:-:-:-:Run NSCF to calculate unoccupied bands because-:-:-:-:-:-:-:-
#-:-:-:-:-metals may not have a gap in the range over which QE calculates-:-:-:-
#-:-:-:-:-:-Reduced number of K_POINTS to reduce computational intensity:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
cat > Au.nscf.in << EOF
 &control
                  calculation = 'nscf'
                 restart_mode = 'from_scratch',
                       prefix = 'Au',
                      tprnfor = .true.
                   pseudo_dir = '$PSEUDO_DIR/',
                       outdir = '$TMP_DIR/'
 /
 &system
                        ibrav = 2,
                    celldm(1) = 5.44951,
                          nat = 1,
                         ntyp = 1,
                     lspinorb = .true.,
                     noncolin = .true.,
      starting_magnetization = 0.0,
                  occupations = 'smearing',
                      degauss = 0.02,
                     smearing = 'mp',
                      ecutwfc = 65,
                      ecutrho = 260,
 /
 &electrons
                  mixing_beta = 0.7,
                     conv_thr = 1.0d-8
 /
ATOMIC_SPECIES
  Au  196.96600  Au.rel-pbesol-n-nc.UPF
ATOMIC_POSITIONS {crystal}
  Au   0.000000000000000   0.000000000000000   0.000000000000000
```

```

K_POINTS crystal
         60
        0.0000000000    0.0000000000    0.0000000000    1.0
        0.0000000000    0.0277777778    0.0277777778    1.0
        0.0000000000    0.0555555556    0.0555555556    1.0
        0.0000000000    0.0833333333    0.0833333333    1.0
        0.0000000000    0.1111111111    0.1111111111    1.0
        0.0000000000    0.1388888889    0.1388888889    1.0
        0.0000000000    0.1666666667    0.1666666667    1.0
        0.0000000000    0.1944444444    0.1944444444    1.0
        0.0000000000    0.2222222222    0.2222222222    1.0
        0.0000000000    0.2500000000    0.2500000000    1.0
        0.0000000000    0.2777777778    0.2777777778    1.0
        0.0000000000    0.3055555556    0.3055555556    1.0
        0.0000000000    0.3333333333    0.3333333333    1.0
        0.0000000000    0.3611111111    0.3611111111    1.0
        0.0000000000    0.3888888889    0.3888888889    1.0
        0.0000000000    0.4166666667    0.4166666667    1.0
        0.0000000000    0.4444444444    0.4444444444    1.0
        0.0000000000    0.4722222222    0.4722222222    1.0
        0.0000000000    0.5000000000    0.5000000000    1.0
        0.0277777778    0.5277777778    0.5000000000    1.0
        0.0555555556    0.5555555556    0.5000000000    1.0
        0.0833333333    0.5833333333    0.5000000000    1.0
        0.1111111111    0.6111111111    0.5000000000    1.0
        0.1388888889    0.6388888889    0.5000000000    1.0
        0.1666666667    0.6666666667    0.5000000000    1.0
        0.1944444444    0.6944444444    0.5000000000    1.0
        0.2222222222    0.7222222222    0.5000000000    1.0
        0.2500000000    0.7500000000    0.5000000000    1.0
        0.2708333333    0.7500000000    0.4791666667    1.0
        0.2916666667    0.7500000000    0.4583333333    1.0
        0.3125000000    0.7500000000    0.4375000000    1.0
        0.3333333333    0.7500000000    0.4166666667    1.0
        0.3541666667    0.7500000000    0.3958333333    1.0
        0.3750000000    0.7500000000    0.3750000000    1.0
        0.3863636364    0.7272727273    0.3863636364    1.0
        0.3977272727    0.7045454545    0.3977272727    1.0
        0.4090909091    0.6818181818    0.4090909091    1.0
        0.4204545455    0.6590909091    0.4204545455    1.0
        0.4318181818    0.6363636364    0.4318181818    1.0
        0.4431818182    0.6136363636    0.4431818182    1.0
        0.4545454545    0.5909090909    0.4545454545    1.0
        0.4659090909    0.5681818182    0.4659090909    1.0
        0.4772727273    0.5454545455    0.4772727273    1.0
        0.4886363636    0.5227272727    0.4886363636    1.0
        0.5000000000    0.5000000000    0.5000000000    1.0
        0.4666666667    0.4666666667    0.4666666667    1.0
        0.4333333333    0.4333333333    0.4333333333    1.0
        0.4000000000    0.4000000000    0.4000000000    1.0
        0.3666666667    0.3666666667    0.3666666667    1.0
        0.3333333333    0.3333333333    0.3333333333    1.0
        0.3000000000    0.3000000000    0.3000000000    1.0
        0.2666666667    0.2666666667    0.2666666667    1.0
        0.2333333333    0.2333333333    0.2333333333    1.0
        0.2000000000    0.2000000000    0.2000000000    1.0
        0.1666666667    0.1666666667    0.1666666667    1.0
        0.1333333333    0.1333333333    0.1333333333    1.0
        0.1000000000    0.1000000000    0.1000000000    1.0
        0.0666666667    0.0666666667    0.0666666667    1.0
        0.0333333333    0.0333333333    0.0333333333    1.0
        0.0000000000    0.0000000000    0.0000000000    1.0

EOF
$ECHO "  running the non-scf calculation for Au with norm-conserving PP...\c"
$PW_COMMAND < Au.nscf.in > Au.nscf.out
check_failure $?
$ECHO " done"

ELAPSED_TIME2=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME2 seconds"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-Begin epsilon.x Calculation-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:Use Norm-Conserving Pseudopotentials for epsilon.x to work-:-:-:-:-
    #-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > Au.eps.in << EOF
 &inputpp
                        outdir = '$TMP_DIR/'
                        prefix = 'Au'
                   calculation = 'eps'
/
&energy_grid
                     smeartype = 'lorentz'
                    intersmear = 2.30d0
                    intrasmear = 0.00d0
                          wmax = 6.5d0
                          wmin = 0.0d0
                            nw = 500
                         shift = 0.0d0
/
EOF
$ECHO "running the permittivity calculation...\c"
$EPS_COMMAND < Au.eps.in > Au.eps.out
check_failure $?
$ECHO "done"

ELAPSED_TIME3=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME3 seconds"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:Create Au_Permittivity.dat-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:Values taken from:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:P. B. Johnson and R. W. Christy. Optical constants of the noble metals,:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-Phys. Rev. B 6, 4370-4379 (1972):-:-:-:-:-:-:-:-:-:-:-:-
#-:-:Values sorted as Wavelength (nm), n, k, energy (eV), real, and imaginary-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
$ECHO "Creating Au_Permittivity.dat from Johnson Christy Data...\c"

cat > Au.dat << EOF

```

```
1.93700000 0.92000000 13.78000000 0.64016520 -189.04200000 25.35520000
1.61000000 0.56000000 11.21000000 0.77018634 -125.35050000 12.55520000
1.39300000 0.43000000 9.51900000 0.89016511 -90.42646100 8.18634000
1.21600000 0.35000000 8.14500000 1.01973684 -66.21852500 5.70150000
1.08800000 0.27000000 7.15000000 1.13970588 -51.04960000 3.86100000
0.98400000 0.22000000 6.35000000 1.26016260 -40.27410000 2.79400000
0.89200000 0.17000000 5.66300000 1.39013453 -32.04066900 1.92542000
0.82110000 0.16000000 5.08300000 1.51016929 -25.81128900 1.62656000
0.75600000 0.14000000 4.54200000 1.64021164 -20.61016400 1.27176000
0.70450000 0.13000000 4.10300000 1.76011356 -16.81770900 1.06678000
0.65950000 0.14000000 3.69700000 1.88021228 -13.64820900 1.03516000
0.61680000 0.21000000 3.27200000 2.01037613 -10.66188400 1.37424000
0.58210000 0.29000000 2.86300000 2.13021818 -8.11266900 1.66054000
0.54860000 0.43000000 2.45500000 2.26029894 -5.84212500 2.11130000
0.52090000 0.62000000 2.08100000 2.38049530 -3.94616100 2.58044000
0.49590000 1.04000000 1.83300000 2.50050413 -2.27828900 3.81264000
0.47140000 1.31000000 1.84900000 2.63046245 -1.70270100 4.84438000
0.45090000 1.38000000 1.91400000 2.75005544 -1.75899600 5.28264000
0.43050000 1.45000000 1.94800000 2.88037166 -1.69220400 5.64920000
0.41330000 1.46000000 1.95800000 3.00024195 -1.70216400 5.71736000
0.39740000 1.47000000 1.95200000 3.12028183 -1.64940400 5.73888000
0.38150000 1.46000000 1.93300000 3.25032765 -1.60488900 5.64436000
0.36790000 1.48000000 1.89500000 3.37048111 -1.40062500 5.60920000
0.35420000 1.50000000 1.86600000 3.50084698 -1.23195600 5.59800000
0.34250000 1.48000000 1.87100000 3.62043796 -1.31024100 5.53816000
0.33150000 1.48000000 1.88300000 3.74057315 -1.35528900 5.57368000
0.32040000 1.54000000 1.89800000 3.87016230 -1.23080400 5.84584000
0.31070000 1.53000000 1.89300000 3.99098809 -1.24254900 5.79258000
0.30090000 1.53000000 1.88900000 4.12097042 -1.22742100 5.78034000
0.29240000 1.49000000 1.87800000 4.24076607 -1.30678400 5.59644000
0.28440000 1.47000000 1.86900000 4.36005626 -1.33226100 5.49486000
0.27610000 1.43000000 1.84700000 4.49112640 -1.36650900 5.28242000
0.26890000 1.38000000 1.80300000 4.61137970 -1.34640900 4.97628000
0.26160000 1.35000000 1.74900000 4.74006116 -1.23650100 4.72230000
0.25510000 1.33000000 1.68800000 4.86083889 -1.08044400 4.49008000
0.24900000 1.33000000 1.63100000 4.97991968 -0.89126100 4.33846000
0.24260000 1.32000000 1.57700000 5.11129431 -0.74452900 4.16328000
0.23710000 1.32000000 1.53600000 5.22986082 -0.61689600 4.05504000
0.23130000 1.30000000 1.49700000 5.36100303 -0.55100900 3.89220000
0.22620000 1.31000000 1.46000000 5.48187445 -0.41550000 3.82520000
0.22140000 1.30000000 1.42700000 5.60072267 -0.34632900 3.71020000
0.21640000 1.30000000 1.38700000 5.73012939 -0.23376900 3.60620000
0.21190000 1.30000000 1.35000000 5.85181689 -0.13250000 3.51000000
0.20730000 1.30000000 1.30400000 5.98166908 -0.01041600 3.39040000
0.20330000 1.33000000 1.27700000 6.09936055 0.13817100 3.39682000
0.19930000 1.33000000 1.25100000 6.22177622 0.20389900 3.32766000
0.19530000 1.34000000 1.22600000 6.34920635 0.29252400 3.28568000
0.19160000 1.32000000 1.20300000 6.47181628 0.29519100 3.17592000
0.18790000 1.28000000 1.18800000 6.59925492 0.22705600 3.04128000

EOF

$ECHO "done"

ELAPSED_TIME4=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME4 seconds"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
```

```
#-:-:-:-:-:-Begin Plotting with GNUPlot (Assuming That It is Installed)-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
if [ "$GP_COMMAND" = "" ]; then
    break
else
cat > gnuplot.tmp <<EOF
#!$GP_COMMAND

set term png size 1920,1080
set xrange [0:6.5]

set out 'epsi_Au.png'
set timestamp
set title 'epsi_Au'
set ylabel 'epsilon'
set xlabel 'Energy (eV)'
set yrange [-50:50]
plot "epsi_Au.dat" using 1:2 title 'epsi_x' ps 0.5, "epsi_Au.dat" using 1:3 title '
    epsi_y' ps 0.5, "epsi_Au.dat" using 1:4 title 'epsi_z' ps 0.5, "Au.dat" using
    4:6 ps 0.5

set out 'epsr_Au.png'
set timestamp
set title 'epsr_Au'
set ylabel 'epsilon'
set xlabel 'Energy (eV)'
set yrange [-50:50]
plot "epsr_Au.dat" using 1:2 title 'epsr_x' ps 0.5, "epsr_Au.dat" using 1:3 title '
    epsr_y' ps 0.5, "epsr_Au.dat" using 1:4 title 'epsr_z' ps 0.5, "Au.dat" using
    4:5 ps 0.5

set out 'eps_combined_Au.png'
set timestamp
set title 'eps_combined_Au'
set ylabel 'epsilon'
set xlabel 'Energy (eV)'
set yrange [-50:50]
plot "epsi_Au.dat" using 1:2 title 'epsi_x' ps 0.5, "epsi_Au.dat" using 1:3 title '
    epsi_y' ps 0.5,  "epsi_Au.dat" using 1:4 title 'epsi_z' ps 0.5, "epsr_Au.dat"
    using 1:2 title 'epsr_x' ps 0.5,    "epsr_Au.dat" using 1:3 title 'epsr_y' ps
    0.5, "epsr_Au.dat" using 1:4 title 'epsr_z' ps 0.5, "Au.dat" using 4:6 title '
    Aui' ps 0.5, "Au.dat" using 4:5 title 'Aur' ps 0.5

set out 'eps_combined_averaged_Au.png'
set timestamp
set title 'eps_combined_averaged_Au'
set ylabel 'epsilon'
set xlabel 'Energy (eV)'
set yrange [-50:50]
plot "epsi_Au.dat" u 1:(($2+$3+$4)/3) ps 0.5, "epsr_Au.dat" u 1:(($2+$3+$4)/3) ps
    0.5, "Au.dat" u 4:5 ps 0.5, "Au.dat" u 4:6 ps 0.5

EOF
$ECHO
$ECHO "  Plotting Real and Imaginary Parts of Calculated Permittivies ...\c"
$GP_COMMAND < gnuplot.tmp
$ECHO " done"
rm gnuplot.tmp
```

```
fi

ELAPSED_TIME5=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME5 seconds"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-Begin Cleaning Temp Directory-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

$ECHO "  cleaning $TMP_DIR...\c"
rm -rf $TMP_DIR/Au.*

$ECHO
$ECHO "$EXAMPLE_DIR: done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:End Elapsed Time Measurement-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

ELAPSED_TIME=$(($SECONDS - $START_TIME))
echo "It has been $ELAPSED_TIME seconds"

echo "Job Completed"
```

6.7 Cif2Cell: create an interactive BASH script for crystal-making

This menu program was partially inspired by an underappreciated YouTube BASH tutorial called "Creating Command Line Menus with Shell Scripts" by theurbanpenguin and the general form of a script like this is discussed at greater length in sections 5.1, and 5.4 of this text. This script has been heavily modified to be based on functions by myself however so that it has increased functionality and is more compatible with the shell program called getopts (discussed in section 5.5 of this text).

This set of menus is superfluous to the point of the program that they command, however I thought that it would be a nice way of wrapping up sever commands into a neat little teaching tool (being the menu program). That being said however, I am generally loathe to remember long commands and I either will use an alias (see section 2.6.2 for more information on the alias command and how to use it) or I'll put all of the commands into a well-formatted menu program like the following.

In this section I show a menu program with several common operations that can be executed easily on a .cif file in the program called Cif2Cell. Cif2Cell is a very nice program that has all sorts of functionality which is beyond the scope of this text. However, I do find some of its commands extremely useful and I'm including them for reference here. The script is broken up into three parts, each of which occupy a separate case in the switch/case list.

In the first case, we use Cif2Cell to generate a crystal that is oriented such that the crystal's (100) direction is parallel to the z-axis of the space. In the second case, we use Cif2Cell to create a crystal that is oriented such that the crystal's (111) direction is parallel to the z-axis of the space. And, in the third case, we create a 2x2x2 supercell with the crystal's (100) direction parallel to the z-axis of the space from a .cif file. I find that these are all common operations which I use frequently. Another program with similar capabilities that I find myself using more often these days than Cif2Cell is named Atomsk and can be seen more starting in section 8 of this text. Please see the below script and how it can be used to run Cif2Cell: it is recommended that you supply a simple .cif file for the program at first just so that you can see more easily what operations the program has performed on the data within the file(s).

```bash
#! /bin/bash

function selection () {
echo -e "\n"          # Add a new line with \n
echo -e "Enter your selection \c" # Suppress a new line with \c

function menu () {
# This block launches an interactive Cif2Cell operations menu
clear
echo "=============================================="
echo "====== Cif2Cell Program Launch Menu ======"
echo "=============================================="
echo "Enter 1 to run z-(100) Cif2Cell on a .cif file for pwscf"
echo "Enter 2 to run z-(111) Cif2Cell on a .cif file for pwscf"
echo "Enter 3 to run z-(100) 2x2x2 supercell Cif2Cell on a .cif file for pwscf"
echo "Enter q to exit the menu"
    selection
read answer_one
case $answer_one in
  1) # Executes the Cif2Cell python program on a .cif file
    cwd=$(pwd) # Store current working directory as cwd
    cd /home/steven/Documents/Programs/cif2cell
    echo -e "Enter the name and file path of the .cif file \c"
    read ciffilename     # Read user input and store as $ciffilename
    ./cif2cell -p pwscf --setup-all --print-symmetry-operations --pwscf-atomic-units --pwscf-pseudostring=.pbe-spn-kjpaw_psl.1.0.0.UPF -f $ciffilename
    mv *.in $cwd
    cd $cwd
    echo -e "Check your current directory for the new .in file"
    ;;
  2) # Executes the Cif2Cell python program on a .cif file
    cwd2=$(pwd) # Store current working directory as cwd2
    cd /home/steven/Documents/Programs/cif2cell
    echo -e "Enter the name and file path of the .cif file \c"
    read ciffilename2    # Read user input, store as $ciffilename2
    ./cif2cell -p pwscf --setup-all --print-symmetry-operations --pwscf-atomic-units --cubic-diagonal-z --pwscf-pseudostring=.pbe-spn-kjpaw_psl.1.0.0.UPF -f $ciffilename2
    mv *.in $cwd2
    cd $cwd2
    echo -e "Check your current directory for the new .in file"
    ;;
  3) # Executes the Cif2Cell python program on a .cif file
    cwd=$(pwd) # Store current working directory as cwd
    cd /home/steven/Documents/Programs/cif2cell
    echo -e "Enter the name and file path of the .cif file \c"
    read ciffilename     # Read user input, store as $ciffilename
    echo -e "Enter the supercell size as [x,y,z]"
    read supercell
    ./cif2cell -p pwscf --setup-all --print-symmetry-operations --pwscf-atomic-units --pwscf-pseudostring=.pbe-spn-kjpaw_psl.1.0.0.UPF -f $ciffilename --no-reduce --supercell=$supercell
    mv *.in $cwd
    cd $cwd
    echo -e "Check your current directory for the new .in file"
    ;;
  q) # Executes the exit command from this menu application
    exit
    ;;
esac # End program launch sub-menu switch case block
read input_one
    ;; # End program launcher

} # end of menu function

while [ 1 ]; do menu; done # Run menu function
```

6.8 Firmi: create Fermi surface .stl files from Quantum Espresso output

Firmi is a delightful piece of code written by David Strubbe (also a co-author of Berkeley GW) which has the capability of turning a Fermi surface file from Quantum Espresso (one with the .bxsf file extension) into a format that is readable by openscad. This is very special because, to the best of my knowledge, the program called XCrySDen is the main one people use for reading .bxsf formatted files. However, for as good as XCrySDen is, it does not appear at the time of this writing to be able to export the Fermi surface files into anything resembling a modern 3D file.

As far as I can tell, Firmi was originally designed 'to prepare Fermi surfaces for 3D printing'. Apparently the most suitable format for the Firmi program author for this purpose was openscad. However, if one is intending to create 3D graphics of the Fermi surface, this format may not be so useful; instead of openscad, I greatly prefer the .stl format for its ease of use and universality. The workaround is to use the openscad tool in the terminal to convert the openscad Fermi surface converted with the Firmi code into the .stl format.

In order to do this, we first need to compile the Firmi code so that we can run a specific program inside Firmi called bxsf2scad.x. You will need a FORTRAN compiler of your choice here but I've never had any problems compiling the code.

Run the following code to get the Firmi program and unpack it:

```
wget https://faculty.ucmerced.edu/dstrubbe/Firmi/firmi_v1.0.tar.gz
tar -xzf firmi_v1.0.tar.gz
cd firmi_v1.0
```

Listing out the contents of the new directory should give the following:

```
MainUsers-iMac:firmi_v1.0 mainuser$ ls
LICENCE        bxsf2scad.f90      marching_cubes_edges.data
Makefile       example_copper     marching_cubes_triangles.data
README         example_lead       polyhedron.f90
```

Run the following to compile Firmi:

```
MainUsers-iMac:firmi_v1.0 mainuser$ make
```

If the compilation was successful, you should have a FORTRAN program called bxsf2scad.x that has been compiled into the directory. Now copy and paste the .bxsf file that you wish to turn into a .scad file into the directory with your bxsf2scad.x program. Change the name of your .bxsf file to be explicitly 'bxsf' e.g. without a file extension. Then in the terminal run the following after making sure that bxsf2scad.x is executable:

```
MainUsers-iMac:firmi_v1.0 mainuser$ ./bxsf2scad.x
```

The resulting file will be called bxsf.scad and may be opened in openscad. Render and export the resulting .scad file in openscad as a .stl file. Import the .stl file into Blender and recalculate normals in edit mode (I do find that recalculating the normals is very important and leads to much cleaner looking rendering results). It is also usually helpful to use Blender's tools to smooth and decimate the mesh to some extent to make sure that it looks decent in rendering.

7 Shell Scripts for DFT Calculations with Exciting

Exciting [6, 7, 8, 9, 10] is a DFT code that, in my opinion, has gotten a lot more interesting since the release of its version code-named Oxygen. It's very...exciting... The name is perhaps apt because its main claim to fame seems to be that it is good at simulating excited states. Exciting.Oxygen has several attractive features like TD-DFT, second harmonic generation (SHG), pump-probe spectroscopy, and Raman spectroscopy as built-ins.

However, it's clever name does actually make it exceedingly difficult to search the internet for help and/or example scripts for use with the program. Additionally, I do find that it is difficult to compile, much more-so than many other codes like VASP, and Quantum Espresso. That being said, the appeal of its new and fun functionality made it worth the effort to compile the pure mpi version for myself to give it a shot to see how it worked. Long story short, I enjoy using Exciting and think that its user base will continue to grow in the coming years, especially if they keep releasing interesting and useful features like they did with their most recent (at the time of this writing) version.

7.1 Compiling Exciting Oxygen

Compiling Exciting has always been tricky in my estimation, however newer packages seem to be slightly more forgiving and come pre-packaged with some libraries which makes things easier for us. The most important thing when it comes to compiling is that you actually have the proper files in the locations that the compilers and the shell program called make are expecting. If you are working on a cluster system, it is equivalently important to have the correct module files loaded. In this case, we need a FORTRAN compiler and a C compiler and in this specific case we will use the Intel tools including ifort as the FORTRAN compiler.

Please see the following terminal session for the compilation of Exciting Oxygen including a list of all module files that were loaded during the compilation:

```
uname@uname:~> ls
exciting.oxygen.tar.gz
uname@uname:~> untar exciting.oxygen.tar.gz
uname@uname:~> ls
exciting  exciting.oxygen.tar.gz
uname@uname:~> cd exciting; ls
uname@uname:~/exciting> ls
COPYING   README.md                build     src        xml
INSTALL   TODO                     docs      test
LICENSE   bin                      external  tools
Makefile  bspline-fortran-licence  species   utilities
uname@uname:~/exciting> module list
Currently Loaded Modulefiles:
  1) modules/3.2.11.4
  2) altd/2.0
  3) darshan/3.2.1
  4) craype-network-aries
  5) craype-haswell
  6) craype-hugepages2M
  7) libfabric/1.8.1
  8) impi/2020
  9) intel/19.0.3.199
 10) craype/2.6.2
 11) cray-mpich/7.7.10
 12) cray-libsci/19.06.1
 13) udreg/2.3.2-7.0.1.1_3.59__g8175d3d.ari
 14) ugni/6.0.14.0-7.0.1.1_7.61__ge78e5b0.ari
 15) pmi/5.0.14
 16) dmapp/7.1.1-7.0.1.1_4.70__g38cf134.ari
 17) gni-headers/5.0.12.0-7.0.1.1_6.44__g3b1768f.ari
 18) xpmem/2.2.20-7.0.1.1_4.27__g0475745.ari
 19) job/2.2.4-7.0.1.1_3.54__g36b56f4.ari
 20) dvs/2.12_2.2.167-7.0.1.1_17.10__ge473d3a2
 21) alps/6.6.58-7.0.1.1_6.28__g437d88db.ari
 22) rca/2.2.20-7.0.1.1_4.72__g8e3fb5b.ari
 23) atp/2.1.3
 24) PrgEnv-intel/6.0.5
 25) nano/2.6.3
uname@uname:~/exciting> cp build/platforms/make.inc.ifort build/make.inc
uname@uname:~/exciting> make mpi
```

At this point, the program will begin to compile and you should see something like the following begin to be written to your terminal:

```
cd build/mpi; make
make[1]: Entering directory '~/exciting/build/mpi'
make -f ../Make.common
make[2]: Entering directory '~/exciting/build/mpi'
../../build/utilities/mkmf -t ./template -f -m Makefile.libbzint -p libbzint.a \
../../src/src_libbzint \
&& make -f Makefile.libbzint libbzint.a
..................................................................... Makefile.
    libbzint is ready.
make[3]: Entering directory '~/exciting/build/mpi'
```

After some time compiling, if there are no errors, then you should see the program exit successfully after copying the Exciting executable into the /bin directory under the new name exciting_purempi. There are several binaries that can be built with Exciting but since the program is resource intensive to run useful simulations, the most practically useful to compile is the mpi version.

```
make[3]: Leaving directory '~/exciting/build/mpi'
cp exciting ../../bin/exciting_purempi
cd ../../bin && ../build/utilities/create_sym_link.sh exciting_purempi
make[2]: Leaving directory '~/exciting/build/mpi'
make[1]: Leaving directory '~/exciting/build/mpi'
```

If one were to create a log file of all that was printed to their terminal (see 4.2 for more on how to do this automatically) we could use a redirect into the wc program (please see section 4.1 for more on this functionality) to count the lines that have been printed to the terminal:

```
uname@uname:~/exciting> wc -l < MAKE.out
1261
```

The amount of time that it will take to compile will vary from system to system; however, in the case of this specific compilation, it took about 20 minutes to complete successfully.

We can check on the executable with the following (ignore the fatal error when trying to execute the program here, this just comes about because the requirements for successfully running an Exciting calculation have not been met in this directory, e.g., input files, etc... do not exist in this directory):

```
uname@uname:~/exciting> wc -l < MAKE.out
1261
uname@uname:~/exciting> cd bin/
uname@uname:~/exciting/bin> ls
exciting_purempi
uname@uname:~/exciting/bin> file exciting_purempi
exciting_purempi: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
    linked, interpreter /lib64/l, BuildID[sha1]=
    bc3bx4fceax134545b966deac56d3f4005e3364, for GNU/Linux 3.2.0, with debug_info,
    not stripped
uname@uname:~/exciting/bin> exciting_purempi
Abort(1091471) on node 0 (rank 0 in comm 0): Fatal error in PMPI_Init: Other MPI
    error, error stack:
MPIR_Init_thread(703).......:
MPID_Init(958)..............:
MPIDI_OFI_mpi_init_hook(883): OFI addrinfo() failed (ofi_init.c:883:
    MPIDI_OFI_mpi_init_hook:No data available)
uname@uname:~/exciting/bin>
```

If you encounter strange errors saying something about a C compiler (or other compiler for that matter) not being able to be found or used, check that your modules have been loaded correctly and/or that your compilers are in the places that Exciting expects.

7.2 Some common error sources in Exciting

As is common with me, I encounter all sorts of errors when I run calculations, especially when I am trying a new-to-me code. Here is a partial list of some error sources and potential solutions that I have found for use with Exciting.Oxygen. I am including these items with the presentation first of the error you may see, and then a way you may attempt to resolve the error.

Error	Potential Resolution
Error(checkmt): muffin-tin spheres overlap	try reducing rmt="..." in the species section of your xml file, or try setting autormt="true" at the end of the structure tag
if a large block of many repeating lines of text shows up in the slurm...out files	try running the following (depending heavily on what computer system you are using): module load impi; module load openmpi; module load gcc; module load cce
if a large number of files with names like EVALFV1-3.OUT are created in the directory	your lattice vectors may be strange or without enough places after the decimal, alternatively your rmt value may be too high (try perhaps 1.725 or 1.200)
Exciting quits without the ability to make .xml files for species types	make sure you've run SETUP-excitingroot.sh and that your environment variables are correct

7.3 Ground state energy of NaCl

The Exciting code website has good documentation and quite a few example scripts for some of its calculations. However, as is my custom, I vastly prefer to have a single script to handle everything automatically for me so that I reduce the chance of forgetting some aspect of a calculation. In the following section, I will explain a script that is used to calculate the ground state energy of a rocksalt NaCl crystal using Exciting.

The following Exciting script is separated into three general sections. In the first section (everything before the creation of the input.xml file for Exciting), we set up what we're going to need for the calculation including several variables that will tell the program what to do during the calculation and the locations of specific executables. For the executables, there are three main ways to compile Exciting and I was testing the various builds for my own projects. So, in order to swap quickly between the different executables, I set up a switch/case block where I can choose which executable I want to use on the fly based on a variable at the beginning of the script. Additionally, the script creates a README file, and a slurm script for queuing the program on a cluster.

Section two is a block of code which creates the input.xml file for Exciting, several important considerations are discussed as follows: The crystal scale is synonymous with the lattice parameter a_0 and is given in Bohr radii (all values in angstroms are converted to Bohr radii by a factor of 1.88973), and I give the basis vectors in terms of their fractional coordinates within the cell. Next, we define the specific species that are in the calculation: e.g., Na, and Cl. Assuming that your environment variables are set up correctly, then Exciting will populate your directory with all of the interatomic potential files that you need for the calculation. Following this, we give Exciting a grid size, the type of interatomic potential that we want to use, the number of empty states (which is an important parameter that you should explore for yourself, and rgkamx which determines the number of basis functions. It is very important here that gmaxvr > 2*gkmax =2*(rgkmax/rmt), for example gmaxvr="14". Also, for reference, rmt is an acronym in Exciting for 'radius muffin tin'.

Finally, in section three, we submit the calculation to a computer. It is very important here to run the SETUP-excitingroot.sh script, otherwise it's very likely you'll encounter a mountain of errors and not have your potential files generated properly which will halt the program. The script then submits the job using a slurm script generated in section one via sbatch and runs a while loop to update the user about the status of the program on the computer which updates on a rolling five second interval in perpetuity (which can be halted at any time with ctrl+c keystroke).

I also like to build in a small functionality where I copy and paste the following line into the terminal:

```
File_Name="exciting_NaCl"; mkdir ${File_Name}; cd ${File_Name}; touch ${File_Name}.
    sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a
    README.txt
```

The trick here is that the terminal will (with just this single line of commands) create a shell script file, make it executable, launch vim (see section 3.7 for more on vim) so that you can copy and paste your script, execute the script, and then use tee to copy everything printed to the terminal into a README file. This is especially useful on a remote system because it's a hassle to make all of the executables by hand and also to try and remember the specifics of each calculation. This small part of the code handles all of that for you. The functionality with the tee command is inspired by some commands that I found online in a comment by Byte Commander on Stack Exchange so thanks to that person for the tee command inspiration functionality here.

Please see the following script:

```
#!/bin/bash

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
```

```bash
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-: Exciting_NaCl.sh -:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# Make sure that all environment variables have been added to the path in ~/.bashrc for help see: http://exciting.wikidot.com/oxygen-tutorial-scripts-and-environment-variables

# Copy and paste the below line into the cluster terminal to make and run the script (paste into vim and save with :wq)
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks to Byte Commander on Stack Exchange)

# File_Name="exciting_NaCl"; mkdir ${File_Name}; cd ${File_Name}; touch ${File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Give the Following Variables -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Job_Name="exciting_test_NaCl" # Give the name you want to apply to all files here

Job_Time="00:10:00"   # Give the run time in hh:mm:ss
Job_Nodes="8"         # Number of nodes that you want to use for the calculation
Job_Queue="debug"     # Give calculation queue (e.g., 'debug' or 'regular')

Description="a test for NaCl" # Please give a short description of the calculation for the README.txt file
Author="Steven E. Bopp, Materials Science & Engineering"

Module_Version="mpi"                # Give the version that exciting was compiled with. Options for Exciting: smp, mpi, or serial
Module_Name="exciting"              # Give the name of the module that you want to load e.g., vasp, lammps, espresso, exciting, etc...
Header="~/uname/codes" # Give the module location for the exciting version. For all exciting versions: ~/uname/codes

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Automated Variables :-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Loading Modules..."; module load impi; module load openmpi; module load gcc; module load cce

echo " Finding Executable Locations for ${Module_Name} ${Module_Version}..."
case $Module_Version in
  mpi)    echo " using exciting_mpi for calculation..."
  Module_Location="${Header}/exciting_mpi/bin/exciting_purempi";   speciespath="${Header}/exciting_mpi/species";       EXCITINGROOT="${Header}/exciting_mpi"    ;;
  smp)    echo " using exciting_smp for calculation..."
  Module_Location="${Header}/exciting_smp/bin/exciting_smp";       speciespath="${Header}/exciting_smp/species";       EXCITINGROOT="${Header}/exciting_smp"    ;;
  serial) echo " using exciting_serial for calculation..."
  Module_Location="${Header}/exciting_serial/bin/exciting_serial"; speciespath="${Header}/exciting_serial/species";    EXCITINGROOT="${Header}/exciting_serial" ;;
                    # Exciting executable location                                # species path for Exciting directory                # Location of Exciting home directory
```

```
esac

Date=$(date '+%d/%m/%Y %H:%M:%S')            # Give date in day/month/year hr/min/sec thanks user1293137 from https://unix.stackexchange.com/

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF
Job Name: ${Job_Name}.sh
This is a ${Module_Name} calculation of the ${System_Name} system to calculate ${
    Description}.
Calculating with ${Job_Nodes} job nodes on the ${Job_Queue} queue by ${Author} on
    $Date.

A transcript of the calculation as seen from the terminal follows:

EOF

echo " Writing file README.txt..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Begin Exciting File Creation :-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running script ${Job_Name}.sh..."
echo " The time is currently $Date "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > ${Job_Name}.sb << EOF
#!/bin/bash
#SBATCH --job-name=$Job_Name
#SBATCH -N ${Job_Nodes}
#SBATCH -C haswell
#SBATCH -q ${Job_Queue}
#SBATCH -t ${Job_Time}

module load impi
module load openmpi
module load gcc
module load cce

srun -n32 -c2 --cpu_bind=cores ${Module_Location}

EOF

echo " Writing input file ${Job_Name}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:- Create input.xml File for Exciting :-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
cat > input.xml << !
<input>

   <title>NaCl</title>

   <structure speciespath="$EXCITINGROOT/species/">

      <crystal scale="10.658">
            <basevect>0.0   0.5   0.5</basevect>
            <basevect>0.5   0.0   0.5</basevect>
            <basevect>0.5   0.5   0.0</basevect>
      </crystal>

      <species speciesfile="Na.xml">
         <atom coord="0.00 0.00 0.00"/>
      </species>

      <species speciesfile="Cl.xml">
         <atom coord="0.50 0.50 0.50"/>
      </species>

   </structure>

   <groundstate
      do="fromscratch"
      rgkmax="7.0"
      ngridk="4 4 4"
      xctype="LDA_PW"
      nempty="10"
      >
   </groundstate>

</input>
!

echo " Writing input file input.xml..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:- Run Exciting Calculation with SBATCH -:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

SETUP-excitingroot.sh                          # Exciting command, builds xml header,
     important for run

Calculation_Location=$(pwd); echo " Calculation Location: ${Calculation_Location}"

echo " Submitting ${Job_Name}.sb via sbatch..."

sbatch ${Job_Name}.sb                          # Submit job to queue

echo " Running on the ${Job_Queue} queue with ${Job_Nodes} nodes"
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Print Queue to the Terminal -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
echo " The time is currently $Date "

echo " Dumping terminal session into README.txt"

echo " Success, End of Script, Running sqs on a 5 Second Loop"

while [ 1 ] ; do sqs; date; sleep 5; done     # continue to update the squeue every 5 seconds
```

7.4 Second harmonic generation (SHG) of a TiN monolayer (relaxed by VASP) with automatic lattice vector conversion to Bohr radii from POSCAR

Second harmonic generation (SHG) is a nonlinear optical effect whereby some radiation (usually monochromatic laser light from a single source) incident on a crystal with nonzero second order susceptibility $\chi^{(2)}$ may be transformed from the fundamental frequency ω to the doubled frequency 2ω. This is a very useful effect to measure when one wants to determine the magnitude of $\chi^{(2)}$ in a system relative to a crystal of known $\chi^{(2)}$ and in many other arenas like the doubling of 1064 nm lasers to 532 nm: an effect that you have probably seen demonstrated in green laser pointers. In the release of Exciting Oxygen, calculation of SHG is a built-in function.

As with other examples of scripts for Exciting, there is reasonable documentation on their website and the following is inspired by some of their tutorials. For in-depth coverage of all of their features, refer to the Exciting-code's website. In the following section, I will show a script that will calculate the SHG spectrum for a monolayer of TiN which has been relaxed using VASP and given 1 nm of vacuum above its surface (this was done by directly manipulating the VASP POSCAR file basis vectors). The following is not guaranteed to give good results as it stands because of the low convergence thresholds as well as the standard choice of a k-points grid which is inefficient for the vacuum (c-axis) direction. These parameters will need to be modified by the user to suit their own system(s) and benchmarking and convergence criteria. Additionally, because TiN has inversion symmetry in the case of bulk TiN with its rocksalt structure, the value of $\chi^{(2)}$ will be very close to zero.

The code below is separated into four main sections. The first section is similar to that of the previous example (where we calculated the ground state energy of NaCl with Exciting in section 7.3) but has some important differences and extends until the comment in the script saying "Create File POSCAR". In this section we define the elements for the input.xml file as variables at the beginning of the script; I find this to be exceedingly useful so that I can 1) check that everything is in order with the calculation without scrolling, which is very helpful in the case that you have many calculations you are working with, and 2) it creates the script as a template which can be used for mostly any two-element systems without lots of tedious tweaks. It's my experience that trying to make many tedious tweaks many times over many files makes my eyes hurt and I end up forgetting things here and there which just cause trouble down the road. I also choose to neglect including a switch/case block (as I did in section 7.3) to choose the executable and other locations in favor of linking just to the Exciting pure mpi executable to save some space for the sake of brevity.

The second section extends from the comment saying "Create file POSCAR" (line 48) to the comment saying "Begin Exciting File Creation" (line 92). In this section I supply the script with a POSCAR file (in this case it is actually a CONTCAR file that I copied from a VASP calculation to relax a (111) monolayer of TiN) and I have the script extract pertinent parameters from the

POSCAR file. All of the heavy lifting here is done under the comment saying "Create lattice vectors from file POSCAR" (between lines 75 and 91). Here, the script uses awk to search through the POSCAR file previously created and find the lattice vectors which I store in 9 variables represented by the following matrix:

$$\text{Å}\begin{bmatrix} a_i & a_j & a_k \\ b_i & b_j & b_k \\ c_i & c_j & c_k \end{bmatrix}_{POSCAR}$$

The script then uses the shell program bc to convert those lattice vectors into Bohr radii (which are necessary for Exciting but I generally use angstroms so I find it cumbersome to do the conversions by hand and instead automate them) which are represented by the following matrix (where Br denotes the Bohr radius):

$$(\tfrac{1.88973Br}{\text{Å}})\text{Å}\begin{bmatrix} a_i & a_j & a_k \\ b_i & b_j & b_k \\ c_i & c_j & c_k \end{bmatrix}_{POSCAR} = \begin{bmatrix} a_{i,Br} & a_{j,Br} & a_{k,Br} \\ b_{i,Br} & b_{j,Br} & b_{k,Br} \\ c_{i,Br} & c_{j,Br} & c_{k,Br} \end{bmatrix}_{Exciting}$$

Because I choose also to have the POSCAR file with direct coordinates, I just opt to copy and paste these values here since it is simpler than searching with awk and setting them as variables and because I frequently change the number and species of atoms in my calculations.

The third section is similar to that of the previous example (where we calculated the ground state energy of NaCl with Exciting section 7.3) but it does have some key differences. This section begins after the comment in the script saying "Create input.xml File for Exciting" (after line 100) and, as the name would suggest, creates the input.xml file. However, this time I opt to define most of the items here as the previously defined variables for the basis vectors converted to Bohr radii, the elements, and an attempt to make the script robust to forgetting to run the SETUP-excitingroot.sh script at the start of the calculation. Additionally, we have a new block at the end of the script (within the <properties> tag) that tells Exciting to calculate the SHG spectrum. Breaking from the Exciting-supplied tutorial here, I include several <chicomp> tags to save time on my calculation because I like to just have things done the way I'd like them to be done the first time I do them (selfish, I know haha). This <chicomp> tag gives cartesian indices x, y, and z for the second order susceptibility $\chi^{(2)}_{x,y,z}(-2\omega,\omega,\omega)$ and you can supply as many of those tags to Exciting as you reasonably want (something that they don't cover in the tutorial).

Finally, I lump together all of chunks of script after the comment "Create SBATCH Script" into section four. Here, we just create the sbatch file for submitting the calculation to the computer, we create a README file with some of the pertinent information about the calculation for future reference, create a Gnuplot script (please see sections 3.8 10.5 for more on Gnuplot) for plotting $\chi^{(2)}_{x,y,z}(-2\omega,\omega,\omega)$ as a function of the spectrum, and then finally submit the job to the computer. Within the Gnuplot script, I also build in the conversion factors from ESU (which are the default units used by Exciting) to nm/V and from eV to nm because those are my unit preferences. This script can be executed after the calculation(s) are completed by issuing the following command:

```
./SHG_Gnuplot.sh
```

Just like before in section 7.3, I built in the small functionality where I copy and paste the following line into the terminal:

```
File_Name="TiN_Monolayer_SHG_Exciting"; mkdir ${File_Name}; cd ${File_Name}; touch ${File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt
```

These commands will have the terminal create a shell script file, make it executable, launch vim (see section 3.7 for more on vim) so that you can copy and paste your script, execute the script, and then use tee to copy everything printed to the terminal into a README file. I recommend using this functionality on remote systems especially for the sake of convenience.

Please see the following script:

```

 #!/bin/bash

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: TiN_Monolayer_SHG_Exciting.sh :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# Make sure that all environment variables have been added to the path in ~/.bashrc for help see: http://exciting.wikidot.com/oxygen-tutorial-scripts-and-environment-variables

# Copy and paste the below line into the cluster terminal to make and run the script (paste into vim and save with :wq)
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks to Byte Commander on Stack Exchange)

# File_Name="TiN_Monolayer_SHG_Exciting"; mkdir ${File_Name}; cd ${File_Name}; touch ${File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-: Give the Following Variables -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Job_Name="TiN_Monolayer_SHG_Exciting" # Give the name you want to apply to all files

Job_Time="00:30:00"   # Give the run time in hh:mm:ss
Job_Nodes="32"        # Give the number of nodes to use for the calculation
Job_Queue="debug"     # Give the calculation queue (e.g., 'debug' or 'regular')

 Description="a calculation of the SHG spectrum in a relaxed monolayer of TiN" # Please give a short description of the calculation for the README.txt file
 Author="Steven E. Bopp, Materials Science & Engineering"

Number_of_Elements="2"  # Number of elements that are in the simulation
Element_1="Ti"          # Give the symbol of the element 1 in your input.xml file
Element_2="N"           # Give the symbol of the element 2 in your input.xml file
System_Name="TiN"       # Give a calculation title for Exciting

RMT="1.7700"            # Give the muffin-tin radius for Exciting to use

Module_Name="exciting"  # Give the name of the module that you want to load
Module_Location="~/uname/codes/exciting_mpi/bin/exciting_purempi"
speciespath="~/uname/codes/exciting_mpi/species"
EXCITINGROOT="~/uname/codes/exciting_mpi"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-: Automated Variables :-:-:-:-:-:-:-:-:-:-:-:-:-:-
```

```
#−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−

Date=$(date '+%d/%m/%Y %H:%M:%S')             # Give date in day/month/year hr/min/
    sec thanks user1293137 from https://unix.stackexchange.com/

echo " Loading Modules..."; module load impi; module load openmpi; module load gcc;
    module load cce

#−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−
#−:−:−:−:−:−:−:−:−:−:−:−:−:−: Create file POSCAR −:−:−:−:−:−:−:−:−:−:−:−:−:−
#−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−

cat > POSCAR << EOF
# Rocksalt TiN oriented X=[1−10] Y=[11−2
   1.00000000000000
     3.2330470929544859     0.0000000000000000     0.0000000000000000
     0.0000000000000000     5.5250037406813357    −0.0036091328639240
     0.0000000000000000    −0.0101628995152523    10.0000000000000000
   Ti    N
     2      2
Selective dynamics
Direct
 −0.0000000000000000  0.0014082830126670  0.0222836581243136   T   T   T
  0.5000000000000000  0.5014082839766533  0.0222836581243136   T   T   T
  0.5000000000000000  0.1652583846179835  0.0482397116489274   T   T   T
 −0.0000000000000000  0.6652583836539973  0.0482397116489274   T   T   T
EOF

echo " Writing file POSCAR..."
echo "  done"

#−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−
#−:−:−:−:−:−:−:−:−:−: Create lattice vectors from file POSCAR :−:−:−:−:−:−:−:−
#−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−:−

ai=$(awk '{if(NR==3) print $1}' POSCAR); aj=$(awk '{if(NR==3) print $2}' POSCAR); ak
    =$(awk '{if(NR==3) print $3}' POSCAR)
bi=$(awk '{if(NR==4) print $1}' POSCAR); bj=$(awk '{if(NR==4) print $2}' POSCAR); bk
    =$(awk '{if(NR==4) print $3}' POSCAR)
ci=$(awk '{if(NR==5) print $1}' POSCAR); cj=$(awk '{if(NR==5) print $2}' POSCAR); ck
    =$(awk '{if(NR==5) print $3}' POSCAR)

echo " Lattice vectors retrieved from POSCAR"
echo $ai $aj $ak
echo $bi $bj $bk
echo $ci $cj $ck

ai_Br=$(echo "scale=10;($ai)*(1.88973)" | bc); aj_Br=$(echo "scale=10;($aj)
    *(1.88973)" | bc); ak_Br=$(echo "scale=10;($ak)*(1.88973)" | bc)
bi_Br=$(echo "scale=10;($bi)*(1.88973)" | bc); bj_Br=$(echo "scale=10;($bj)
    *(1.88973)" | bc); bk_Br=$(echo "scale=10;($bk)*(1.88973)" | bc)
ci_Br=$(echo "scale=10;($ci)*(1.88973)" | bc); cj_Br=$(echo "scale=10;($cj)
    *(1.88973)" | bc); ck_Br=$(echo "scale=10;($ck)*(1.88973)" | bc)

echo " Lattice vectors converted to Bohr radii"
echo $ai_Br $aj_Br $ak_Br
echo $bi_Br $bj_Br $bk_Br
echo $ci_Br $cj_Br $ck_Br
```

```

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Begin Exciting File Creation :-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running script ${Job_Name}.sh..."
echo " The time is currently $Date "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:- Create input.xml File for Exciting :-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# rmt = radius muffin tim
# Groundstate: It's important that: gmaxvr > 2*gkmax =2*(rgkmax/rmt), for example
    gmaxvr="14"
# All units are atomic, e.g., make sure Bohr radii instead of angstroms, etc...

cat > input.xml << !
<input>

   <title>TiN SHG</title>

   <structure speciespath="$EXCITINGROOT/species" autormt="true">
      <crystal>
      <basevect>${ai_Br} ${aj_Br} ${ak_Br}</basevect>
      <basevect>${bi_Br} ${bj_Br} ${bk_Br}</basevect>
      <basevect>${ci_Br} ${cj_Br} ${ck_Br}</basevect>
      </crystal>
      <species chemicalSymbol="${Element_1}" speciesfile="${Element_1}.xml" rmt="${
    RMT}">
         <atom coord="-0.0000000000000000 0.0014082830126670 0.0222836581243136"/>
         <atom coord=" 0.5000000000000000 0.5014082839766533 0.0222836581243136"/>
      </species>
      <species chemicalSymbol="${Element_2}" speciesfile="${Element_2}.xml" rmt="${
    RMT}">
         <atom coord=" 0.5000000000000000 0.1652583846179835 0.0482397116489274"/>
         <atom coord="-0.0000000000000000 0.6652583836539973 0.0482397116489274"/>
      </species>
   </structure>

   <groundstate
      do="fromscratch"
      rgkmax="6.0"
      ngridk="8 8 8"
      xctype="LDA_PW"
      nempty="10"
      >
   </groundstate>

   <properties>
      <momentummatrix/>
      <shg
         wmax="0.3"
         wgrid="400"
         swidth="0.004"
         etol="1.d-4"
         scissor="0.0423"
         tevout="true"
```

```
        >
        <chicomp>1 2 3</chicomp>
        <chicomp>1 1 2</chicomp>
        <chicomp>2 2 3</chicomp>
        <chicomp>2 1 3</chicomp>
     </shg>
  </properties>

</input>
!

echo " Writing input file input.xml..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > ${Job_Name}.sb << EOF
#!/bin/bash
#SBATCH --job-name=$Job_Name
#SBATCH -N ${Job_Nodes}
#SBATCH -C haswell
#SBATCH -q ${Job_Queue}
#SBATCH -t ${Job_Time}

srun -n32 -c2 --cpu_bind=cores ${Module_Location}

EOF

echo " Writing input file ${Job_Name}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF
Job Name: ${Job_Name}.sh
This is a ${Module_Name} calculation of the ${System_Name} system to calculate ${
    Description}.
Calculating with ${Number_of_Elements} element(s): ${Element_1}, and ${Element_2}
    for ${Job_Time} with ${Job_Nodes} job nodes on the ${Job_Queue} queue.
Calculated by ${Author} on $Date.

A transcript of the calculation as seen from the terminal follows:

EOF

echo " Writing file README.txt..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-: Create file SHG_Gnuplot.sh -:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# **VERY IMPORTANT** EOF is quoted so that $1, $2, etc... will be printed in the
    file (thanks to dogbane from StackOverflow)
```

```
# This GNUPlot script will convert eV to nm via hv = 1240 eV*nm, and also covert esu
     units to nm/V with 1 nm/V = 0.00431778929 esu

cat > SHG_Gnuplot.sh << "EOF"
gnuplot
set terminal svg enhanced
set title "TiN X^(2)"
set xlabel 'Wavelength (nm)'
set ylabel "X^(2)"
set xrange [150:1000]; # in principle this range can go past ~60000 nm
set out 'TiN_CHI-123.svg'
p 'TiN_CHI-123.dat' u (1240/$1):($2*0.00431778929) title 'Real' w lines, 'TiN_CHI
    -123.dat' u (1240/$1):($3*0.00431778929) title 'Imag' w lines, 'TiN_CHI-123.dat'
     u (1240/$1):($4*0.00431778929) title 'Modulus' w lines
EOF

chmod +x SHG_Gnuplot.sh # Make the script executable with +x

echo " Writing file SHG_Gnuplot.sh..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-: Run Exciting Calculation with SBATCH -:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

SETUP-excitingroot.sh   # Exciting command, builds xml header, important for run

Calculation_Location=$(pwd); echo " Calculation Location: ${Calculation_Location}"

echo " Submitting ${Job_Name}.sb via sbatch..."

sbatch ${Job_Name}.sb   # Submit job to queue

echo " Running on the ${Job_Queue} queue with ${Job_Nodes} nodes"
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Print Queue to the Terminal -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " The time is currently $Date "

echo " Dumping terminal session into README.txt"

echo " Success, End of Script, Running sqs on a 5 Second Loop"

while [ 1 ] ; do sqs; date; sleep 5; done # update the squeue every 5 seconds
```

8 Create Crystals and Heterostructures in the Shell with Atomsk

Atomsk [11] is a free and open source software that is very useful for creating and manipulating crystallographic files of all sorts through the command line. The Atomsk website has a large number of useful tutorials and a great reference of all the commands and options that you might need for creating and manipulating your files. In the following sections, I cover several use cases of Atomsk as well as the the input and output of various commands that I find to be useful for my research.

8.1 Installing Atomsk on Linux

Installation of Atomsk, I find, is most useful when it can be added easily to the path. If it can't then there will be a lot of functionality missing from the install. Overall, I'd say its easiest (on Linux) to install Atomsk from a .deb package. This can be done simply (assuming that you are in the same directory as the package) by using the following command (edited for your personal system and file name/location):

```
sudo dpkg -i /home/ubuntu-budgie/Desktop/atomsk_b0.11_amd64.deb
```

This will get you most of the way to a complete install of Atomsk, however some of the packages that Atomsk depends on may not be installed (this will be apparent if you get a warning from the terminal saying something to the effect of 'Atomsk depends on ...; however: package ... is not installed ...' At the time of this writing, this minor issue can be fixed easily by running the following command in the terminal:

```
sudo apt install -f
```

After this operation completes, you should be able to launch Atomsk from the terminal in interactive mode by using the following command:

```
atomsk
```

Running Atomsk like this will result in something like the following being printed to your terminal screen:

```
uname@ubuntu:~/Desktop$ atomsk
 ___________________________________________________
|              ___________                          |
|     o---o    A T O M S K                          |
|    o---o|    Version Beta 0.11                    |
|    |   |o    (C) 2010 Pierre Hirel                |
|    o---o     https://atomsk.univ-lille.fr         |
|___________________________________________________|
*** Working out of office hours? You should sleep sometimes. :-)
>>> Atomsk is a free, Open Source software.
    To learn more, enter 'license'.
>>> Atomsk command-line interpreter:
..> Type "help" for a summary of commands.

uname@atomsk:Desktop>
```

However, I almost never run Atomsk this way and will not throughout the remainder of this text. Most often, it is easiest to run Atomsk by issuing the atomsk command followed by its options all in one command within the terminal.

8.2 Compiling Atomsk from its source on Linux

In the case that you do not want to use a package like .deb, you can compile Atomsk from its source code. And, very pleasantly, compiling Atomsk is thankfully pretty straightforward! Depending on your system, the compilation may be a little bit more of a trick than just running 'make atomsk' but shouldn't be too complicated. For this example, we are compiling with the Intel ifort compiler so we will need to select a specific make file that will utilize ifort. Please see the following terminal session regarding the compilation of Atomsk in Linux:

```
uname@uname:~atomsk> ls
atomsk_b0.11.tar.gz
uname@uname:~atomsk> untar atomsk_b0.11.tar.gz
uname@uname:~atomsk> atomsk_0.11  atomsk_b0.11.tar.gz
uname@uname:~atomsk> cd atomsk_0.11
uname@uname:~atomsk/atomsk_0.11> ls
CHANGELOG  LICENSE  README  doc  etc  examples  man  src  tools
uname@uname:~atomsk/atomsk_0.11> cd src/
uname@uname:~atomsk/atomsk_0.11/src> ls
 Makefile  Makefile.g95  Makefile.gfomp  Makefile.i686  Makefile.ifort  Makefile.
    local  Makefile.macos  Makefile.serial  Makefile.static  Makefile.windows  OBJ
    atomsk.f90  include  input  modes  options  output
uname@uname:~atomsk/atomsk_0.11/src> make -f Makefile.ifort atomsk
```

At this point the compilation should begin and something analogous to the following should begin to be printed to the terminal:

```
      o---o    ___________
    o---o|     A T O M S K
    |   |o
    o---o       Version 0.11

mkdir -p OBJ
make -j1 -C include
make[1]: Entering directory '~atomsk/atomsk_0.11/src/include'
```

And after some time, the compilation should conclude successfully with the following message:

```
make[1]: Leaving directory '/global/u2/u/uname/codes/sources/test_atomsk/atomsk_0
    .11/src/modes'
 ifort -O2 -DOPENMP -qopenmp -module ../OBJ -funroll-loops -cpp -o atomsk OBJ/*.o
    atomsk.f90 -I OBJ -L /opt/intel/lib/intel64/ -L /opt/intel/mkl/lib/intel64/ -
    lmkl_intel_lp64 -lmkl_intel_thread -lmkl_lapack95_lp64 -lmkl_core -liomp5

    \o/ Compilation was successful!

    <i> To install Atomsk system-wide, you may now run:
          sudo make install
```

Finally, we can check that the executable has been compiled correctly:

```
uname@uname:~atomsk/atomsk_0.11/src> ls
 Makefile  Makefile.g95  Makefile.gfomp  Makefile.i686  Makefile.ifort  Makefile.
    local  Makefile.macos  Makefile.serial  Makefile.static  Makefile.windows  OBJ
    atomsk  atomsk.f90  include  input  modes  options  output
uname@uname:~atomsk/atomsk_0.11/src> file atomsk
atomsk: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
    interpreter /lib64/l, BuildID[sha1]=36of612543fc9e2c281e635f33115f74da6xz4j2,
    for GNU/Linux 3.2.0, with debug_info, not stripped
```

Running the sudo make install command for Atomsk does add increased functionality, however you can get away with some of the functionality without running that command when you are not able to access administrator privileges.

8.3 Creating simple structures

With Atomsk, it is very straightforward to create all manner of structures with all sorts of formats. One of my most commonly used formats (for sake of the ease with which they can be manipulated, and of course that they're directly compatible with VASP) is the VASP POSCAR format. Giving the option -vasp at the end of an Atomsk command will format the output as a POSCAR file.

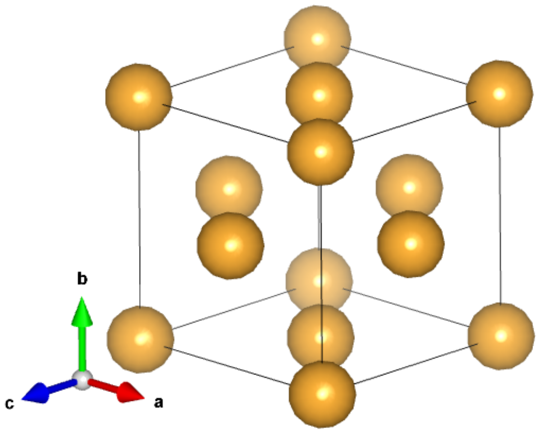
The following example is one of the quickest I can imagine for Atomsk, to create a face centered cubic (FCC) Au crystal with lattice parameter $a_0 = 4.08$Å which is done using the –create option as well as the -frac option which formats the output in fractional coordinates.

```
uname@ubuntu:~/Desktop$ atomsk --create fcc 4.08 Au -frac vasp
 ___________________________________________________
|              ___________                          |
|     o---o    A T O M S K                          |
|    o---o|    Version Beta 0.11                    |
|    |   |o    (C) 2010 Pierre Hirel                |
|    o---o     https://atomsk.univ-lille.fr         |
|___________________________________________________|
 *** Working out of office hours? You should sleep sometimes. :-)
 >>> Creating system:
 ..> Fcc Au oriented X=[100] Y=[010] Z=[001].
 ..> System was successfully created.
 >>> Converting to fractional coordinates...
 ..> Coordinates were reduced.
 >>> Writing output file(s) (4 atoms):
 ..> Successfully wrote POSCAR file: POSCAR
 \o/ Program terminated successfully!
     Total time: 1.066 s.; CPU time: 0.022 s.
```

The plain text output of this command will look like the following POSCAR file:

```
# Fcc Au oriented X=[100] Y=[010] Z=[001].
1.000000
      4.08000000        0.00000000        0.00000000
      0.00000000        4.08000000        0.00000000
      0.00000000        0.00000000        4.08000000
 Au
     4
Direct
      0.00000000        0.00000000        0.00000000
      0.50000000        0.50000000        0.00000000
      0.00000000        0.50000000        0.50000000
      0.50000000        0.00000000        0.50000000
```

This format of the POSCAR file makes sense because we a single atomic species, and the FCC structure for a single species should have four atoms per unit cell (assuming a single atom basis). So, the crystal, if we visualize it in a software like VESTA, will look like the following:



Atomsk can make all sorts of different crystal systems. We can use the following example to create a hexagonal close packed (HCP) crystal of Zr with lattice parameters $a = 3.232$Å, and $c = 5.147$Å. This time, instead of fractional coordinates, we leave the program to default into providing cartesian coordinates.

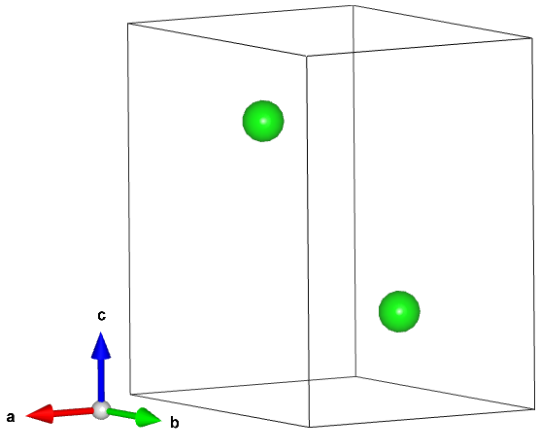
```
atomsk --create hcp 3.232 5.147 Zr vasp
  ___________________________________________________________
 |                 ______________                            |
 |      o---o      A T O M S K                               |
 |    o---o|       Version Beta 0.11                         |
 |    |   |o       (C) 2010 Pierre Hirel                     |
 |    o---o        https://atomsk.univ-lille.fr              |
 |___________________________________________________________|
 *** Working out of office hours? You should sleep sometimes. :-)
 >>> Creating system:
 ..> Hcp Zr with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
 ..> System was successfully created.
 >>> Writing output file(s) (2 atoms):
 ..> Successfully wrote POSCAR file: POSCAR
 \o/ Program terminated successfully!
     Total time: 0.999 s.; CPU time: 0.002 s.
```

Similarly, we can use Atomsk to make an HCP Ti crystal with lattice parameters $a = 2.95111$Å, and $c = 4.68433$Å that will be output as a VASP POSCAR file with fractional coordinates using the following command:

The plain text output of this command will look like the following POSCAR file:

```
# Zr2
1.000000
       3.23923191          0.00000000          0.00000000
      -1.61961595          2.80525712          0.00000000
       0.00000000          0.00000000          5.17222000
 Zr
      2
Direct
       0.66666667          0.33333333          0.75000000
       0.33333333          0.66666667          0.25000000
```

And the crystal, if we visualize it in a software like VESTA, will look like the following:

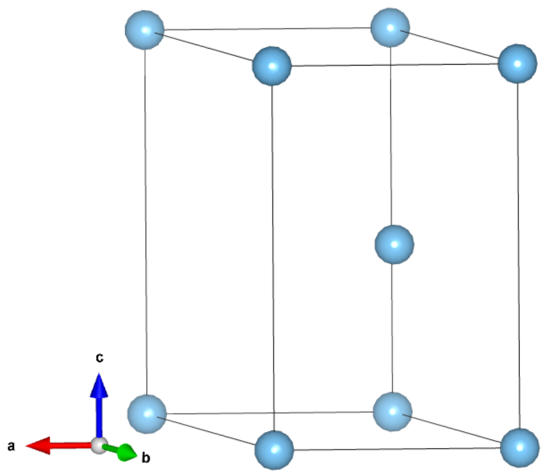


```
uname@ubuntu:~/Desktop$ atomsk --create hcp 2.95111 4.68433 Ti -frac vasp
 ___________________________________________________
|              ___________                          |
|     o---o    A T O M S K                          |
|   o---o|     Version Beta 0.11                    |
|   |   |o     (C) 2010 Pierre Hirel                |
|   o---o      https://atomsk.univ-lille.fr         |
|___________________________________________________|
*** Working out of office hours? You should sleep sometimes. :-)
>>> Creating system:
..> Hcp Ti with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
..> System was successfully created.
>>> Converting to fractional coordinates...
..> Coordinates were reduced.
>>> Writing output file(s) (2 atoms):
..> Successfully wrote POSCAR file: POSCAR
\o/ Program terminated successfully!
    Total time: 1.367 s.; CPU time: 0.018 s.
```

The plain text output of this command (this time in fractional coordinates, in contrast to the previous example for Zr) will look like the following POSCAR file:

```
# Hcp Ti with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
1.000000
      2.95111000       0.00000000       0.00000000
     -1.47555500       2.55573623       0.00000000
      0.00000000       0.00000000       4.68433000
 Ti
     2
Cartesian
      0.00000000       0.00000000       0.00000000
      0.00000000       1.70382415       2.34216500
```

And the crystal, if we visualize it in a software like VESTA, will look like the following:



The wurtzite structure can be easily handled by Atomsk as well. We can create a crystal of aluminum nitride in the wurtzite structure with lattice parameters $a = 3.1117$Å, and $c = 4.68433$Å that's output as a fractional coordinate VASP POSCAR file using the following command:

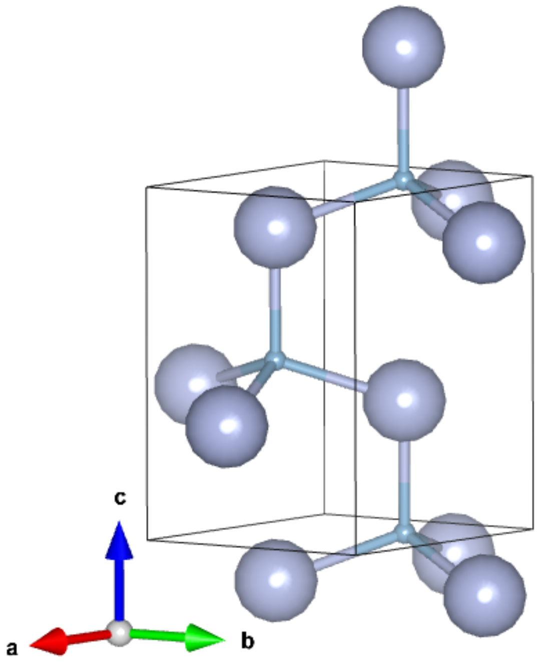
```
uname@ubuntu:~/Desktop$ atomsk --create wurtzite 3.1117 4.9778 Al N -frac vasp
  ___________________________________________________
 |              ___________                          |
 |     o---o    A T O M S K                          |
 |   o---o|     Version Beta 0.11                    |
 |   |   |o     (C) 2010 Pierre Hirel                |
 |   o---o      https://atomsk.univ-lille.fr         |
 |___________________________________________________|
 *** Working out of office hours? You should sleep sometimes. :-)
>>> Creating system:
 ..> AlN with wurtzite structure with box vectors H1=[2-1-10], H2=[-12-10], H3
    =[0001].
 ..> System was successfully created.
>>> Converting to fractional coordinates...
 ..> Coordinates were reduced.
>>> Writing output file(s) (4 atoms):
 ..> Successfully wrote POSCAR file: POSCAR
 \o/ Program terminated successfully!
     Total time: 2.683 s.; CPU time: 0.017 s.
```

The plain text output of this command (again in fractional coordinates) will look like the following POSCAR file:

```
# AlN with wurtzite structure with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
1.000000
      3.11170000         0.00000000         0.00000000
     -1.55585000         2.69481125         0.00000000
      0.00000000         0.00000000         4.97780000
 Al   N
     2         2
Direct
      0.33333333         0.66666667         0.00000000
      0.66666667         0.33333333         0.50000000
```

```
        0.33333333          0.66666667          0.37500000
        0.66666667          0.33333333          0.87500000
```

And the crystal, if we visualize it in a software like VESTA, will look like the following (Al atoms are greenish-blue, N atoms are in gray in this figure):



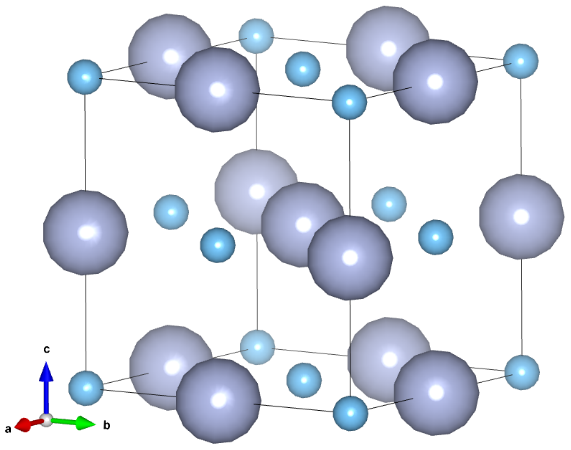
Finally, and perhaps most commonly for my own research at the very least is the rocksalt structure. Atomsk will handle this very nicely for a two-species crystal. Following is the command to create a TiN crystal with lattice parameters $a_0 = 4.241$Å using Atomsk:

```
uname@ubuntu:~/Desktop$ atomsk --create rocksalt 4.241 Ti N -frac vasp
 ___________________________________________________
|              ___________                          |
|     o---o    A T O M S K                          |
|    o---o|    Version Beta 0.11                    |
|    |   |o    (C) 2010 Pierre Hirel                |
|    o---o     https://atomsk.univ-lille.fr         |
|___________________________________________________|
 *** Working out of office hours? You should sleep sometimes. :-)
>>> Creating system:
..> Rocksalt TiN oriented X=[100] Y=[010] Z=[001].
..> System was successfully created.
>>> Converting to fractional coordinates...
..> Coordinates were reduced.
>>> Writing output file(s) (8 atoms):
..> Successfully wrote POSCAR file: POSCAR
\o/ Program terminated successfully!
    Total time: 1.300 s.; CPU time: 0.016 s.
```

The plain text output of this command (again in fractional coordinates) will look like the following POSCAR file:

```
# Rocksalt TiN oriented X=[100] Y=[010] Z=[001].
1.000000
       4.24100000          0.00000000          0.00000000
       0.00000000          4.24100000          0.00000000
       0.00000000          0.00000000          4.24100000
 Ti   N
      4          4
Direct
       0.00000000          0.00000000          0.00000000
       0.50000000          0.50000000          0.00000000
       0.00000000          0.50000000          0.50000000
       0.50000000          0.00000000          0.50000000
       0.50000000          0.00000000          0.00000000
       0.00000000          0.50000000          0.00000000
       0.00000000          0.00000000          0.50000000
       0.50000000          0.50000000          0.50000000
```

And the crystal, if we visualize it in a software like VESTA, will look like the following (Ti atoms are blue, N atoms are in gray in this figure):



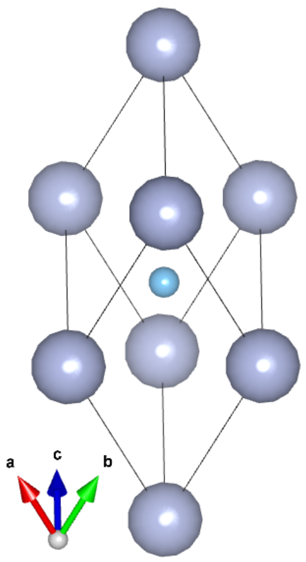
Beware however that sometimes the crystals that Atomsk will generate using the above commands will not be the most economical retelling of the crystallographic information. In fact, the most economical retelling of a crystal is known as the primitive cell and can usually be found easily on repositories of crystallographic information like the materials project.

For example, the primitive cell of TiN can be represented as the following (in VASP POSCAR format):

```
# Ti1 N1
1.000000
       3.00770274          0.00000000          0.00000000
       1.50385137          2.60474698          0.00000000
       1.50385137          0.86824899          2.45577900
 Ti   N
      1          1
```

```
Direct
        0.50000000          0.50000000          0.50000000
        0.00000000          0.00000000          0.00000000
```

And can be visualized as follows with Ti atoms being blue, and N atoms being in gray:



As is plain to see, the primitive cell of TiN has only 2 atoms as compared to the eight atoms of the cell which Atomsk generated from the earlier command. This primitive cell can represent a substantial savings of computational 'expense' compared to the non-primitive cell since it has many fewer atoms. It is therefore useful to consider carefully in what situations you can and cannot use the primitive cell to reduce computational intensity of a crystallographic computation.

8.4 Creation of oriented and duplicated crystals

The goal of simulations is to create as accurate a retelling of a system as is reasonable within the bounds of reasonably economical computational expense. An important component of that the is to supply for the simulation with initial conditions that most accurately retell the specifications of the system(s) you are trying to simulate.

Crystals will frequently have preferred orientations and these are usually kinetically driven and highly dependent on the growth conditions, the substrate onto which the crystal was grown, and other factors like temperature. For that reason, we need to consider the crystallographic orientation which is most thermodynamically reasonable to exist in the real-world counterpart of a simulation. Of course, how can you expect a simulation to give reasonable results if the simulation's conditions are unreasonable?

An example of this that is of particular interest to my own research is the crystallographic growth orientation(s) of Au, and TiN. In many cases, it turns out that Au will favor growing (111) facets, the same usually holds true for TiN. However, TiN has lots of variability in what orientation will grow; for example, the (200) direction may be more favorable on substrates like MgO, whereas the (111) direction may be more favorable on substrates like Al_2O_3. Suffice it to say, it's important to match the crystallographic orientation that is most likely to grow in reality to the orientation that you include in your simulation because the properties of different crystallographic facets in some systems can be wildly different.

Atomsk can handle the creation of oriented crystals and can re-orient crystals based on a series of three vectors that you supply in the form of Miller indices. A review of Miller indices is beyond the scope of this text but generally they are just a set of indices h, k, and l which denote a family of parallel lattice planes in reciprocal space that intersect with the lattice's basis vectors at coordinates of $1/h$, $1/k$, and $1/l$. For the transformation of one orientation to another, Atomsk needs to be supplied a set of three vectors that are orthogonal to each other. There are obviously an infinite number of these; however, since a very common transformation is from the (100) direction to the (111) direction, we will consider that most carefully.

It is worthy of note here that (as given by the YouTube channel Nickel and Copper on several occasions) there is a transformation matrix from (100) to (111). This is very useful if you are trying to re-orient crystals using a program like VESTA. Even though the specifics of transformation matrices are beyond the scope of what we need to go deep into here, I will supply the matrix so that you can more clearly visualize (if that's how your brain works) the transformation from (100) to (111):

$$\begin{bmatrix} 0.5 & 0.5 & 1 \\ 0.5 & -0.5 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

We can create a (111)-oriented crystal (that is the (111) direction will be aligned with the z-axis of the space) of a structure like ZrN using the following command in Atomsk (reproduced here without all of the text printed to the terminal for brevity):

```
uname@ubuntu:~/Desktop$ atomsk --create rocksalt 4.5675 Zr N orient [1-10] [11-2]
    [111] ZrN.cfg
```

Readers can verify for themselves that the vectors $[1-10]$, $[11-2]$, and $[111]$ are orthogonal. The order of these vectors is important and is given as the first vector will be oriented parallel to the x-axis, the second vector will be oriented parallel to the y-axis, and the third vector will be oriented parallel to the z-axis. In this way, the crystal we just created is ZrN with its (111) direction parallel to the z-axis. In this case, the crystal was saved as a .cfg file. To convert that file to VASP POSCAR format we can issue the following command:

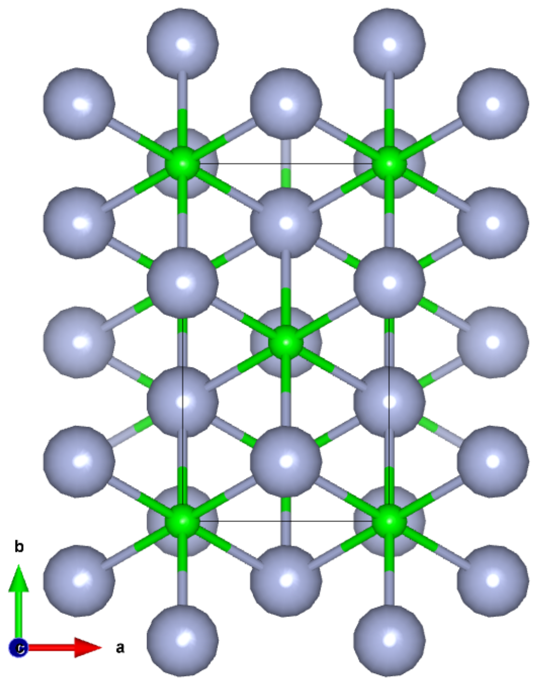
```
uname@ubuntu:~/Desktop$ atomsk ZrN.cfg vasp
```

The resulting POSCAR file will look like the following:

```
# Rocksalt ZrN oriented X=[1-10] Y=[11-2] Z=[111].
1.000000
       3.29334983         0.00000000         0.00000000
       0.00000000         5.70424924         0.00000000
       0.00000000         0.00000000         8.06702664
 Zr   N
      6          6
Cartesian
       0.00000000         0.00000000         0.00000000
       0.00000000         3.80283285         5.37801779
```

```
        0.00000000          1.90141639          2.68900885
        1.64667491          4.75354101          2.68900885
        1.64667491          0.95070823          5.37801779
        1.64667491          2.85212462          0.00000000
        1.64667491          0.95070823          1.34450447
        1.64667491          4.75354101          6.72252217
        1.64667491          2.85212462          4.03351332
        0.00000000          1.90141639          6.72252217
        0.00000000          3.80283285          1.34450447
        0.00000000          0.00000000          4.03351332
```

We can visualize the c-axis projection of this crystal with VESTA as the following:



Another feature that Atomsk has is to create supercells in a crystal with the -duplicate option. The -duplicate option needs to be followed by three integers that correspond respectively to the x, y, and z directions in the crystal. For example, we can create a TiN crystal with its (111) direction oriented along the z-axis, and then duplicate it along an axis to create a supercell:

First, create a (111)-oriented TiN crystal having the rocksalt structure with the following command:

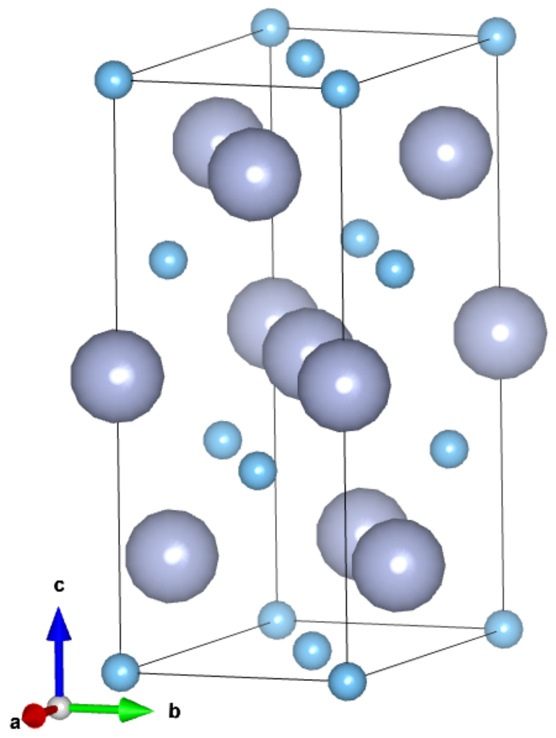
```
uname@ubuntu:~/Desktop$ atomsk --create rocksalt 4.241 Ti N orient [11-2] [1-10]
    [111] vasp
```

This will generate the following POSCAR file:

```
# Rocksalt TiN oriented X=[11-2] Y=[1-10] Z=[111].
1.000000
        5.18679453          0.00000000          0.00000000
```

```
      0.00000000         2.99459722         0.00000000
      0.00000000         0.00000000         7.33523517
 Ti   N
     6         6
Cartesian
      0.00000000         0.00000000         0.00000000
      3.45786302         0.00000000         4.89015678
      1.72893151         0.00000000         2.44507839
      4.32232878         1.49729861         2.44507839
      0.86446576         1.49729861         4.89015678
      2.59339727         1.49729861         0.00000000
      0.86446576         1.49729861         1.22253920
      4.32232878         1.49729861         6.11269598
      2.59339727         1.49729861         3.66761759
      1.72893151         0.00000000         6.11269598
      3.45786302         0.00000000         1.22253920
      0.00000000         0.00000000         3.66761759
```

Which can be visualized as:



Next, we can then duplicate this crystal (since the last command will have saved the crystal into a file named POSCAR) with the following command. Reader make note of several things here. First, Atomsk needs you to tell it which file named POSCAR you want to use, for us the first file it lists is also the file that we want to use, enter 1 into the terminal. Next it asks if we want to overwrite the file named POSCAR since we are trying to create another POSCAR file while one already exists in the same place; select yes because we are trying to modify the POSCAR file and will no longer need the original. Finally, Atomsk asks whether we want to 'pack' atoms so that the species are continuous. This is a quirk of VASP and in its POSCAR files, it needs all of the species to be contiguous since the only way it knows what species are what in the atomic coordinates is the two lines after the basis vectors which give the species in the crystal as well as the number of atoms in the simulation belonging to each individual specie. Please see the following example:

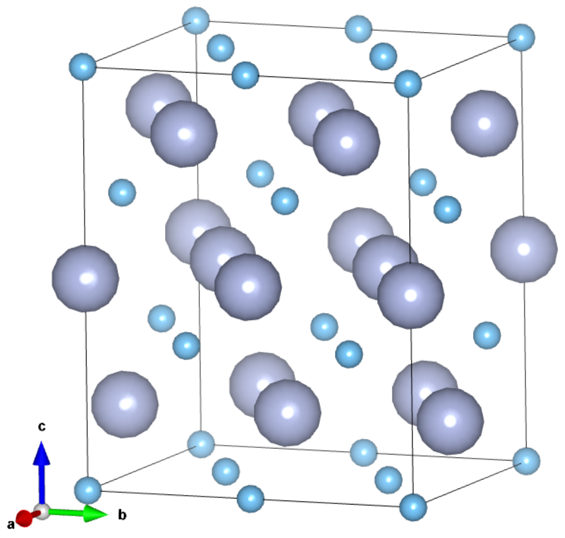
```
uname@ubuntu:~/Desktop$ atomsk POSCAR -duplicate 1 2 1 POSCAR
  ___________________________________________________
 |              ___________                          |
 |     o---o    A T O M S K                          |
 |   o---o|     Version Beta-0.11.1                  |
 |   |   |o     (C) 2010 Pierre Hirel                |
 |   o---o      https://atomsk.univ-lille.fr         |
 |___________________________________________________|
 /!\ Both files exist: POSCAR and POSCAR
 <?> Please indicate which one you want to use as input file:
     1- POSCAR
     2- POSCAR
1
 >>> Opening the input file: POSCAR
 ..> Input file was read successfully (12 atoms).
 >>> Duplicating the system: 1 x 2 x 1
 ..> System was successfully duplicated (24 atoms).
 >>> Writing output file(s) (24 atoms):
 <?> This file already exists: POSCAR
     Do you want to overwrite it (y/n) (Y=overwrite all)?
y
 ..> OK, I will overwrite POSCAR
 /!\ WARNING: atom species are not contiguous. Do you want to pack them? (y/n)
     (this will affect only the POSCAR file)
y
 ..> Atom species were packed: Zr, N.
     Check that this is consistent with the POTCAR file.
 ..> Successfully wrote POSCAR file: POSCAR
 \o/ Program terminated successfully!
     Total time: 11.807 s.; CPU time: 0.005 s.
  _____________________________________________
 |  /!\ WARNINGS: 1                            |
 |_____________________________________________|
```

The resulting POSCAR file will look like the following (note there are now double the number of atoms in the crystal but that the stoichiometry has been maintained):

```
# Rocksalt TiN oriented X=[11-2] Y=[1-10] Z=[111].
1.000000
      5.18679453        0.00000000        0.00000000
      0.00000000        5.98919444        0.00000000
      0.00000000        0.00000000        7.33523517
 Ti   N
    12       12
Cartesian
      0.00000000        0.00000000        0.00000000
      3.45786302        0.00000000        4.89015678
      1.72893151        0.00000000        2.44507839
      4.32232878        1.49729861        2.44507839
      0.86446576        1.49729861        4.89015678
      2.59339727        1.49729861        0.00000000
```

```
        0.00000000          2.99459722          0.00000000
        3.45786302          2.99459722          4.89015678
        1.72893151          2.99459722          2.44507839
        4.32232878          4.49189583          2.44507839
        0.86446576          4.49189583          4.89015678
        2.59339727          4.49189583          0.00000000
        0.86446576          1.49729861          1.22253920
        4.32232878          1.49729861          6.11269598
        2.59339727          1.49729861          3.66761759
        1.72893151          0.00000000          6.11269598
        3.45786302          0.00000000          1.22253920
        0.00000000          0.00000000          3.66761759
        0.86446576          4.49189583          1.22253920
        4.32232878          4.49189583          6.11269598
        2.59339727          4.49189583          3.66761759
        1.72893151          2.99459722          6.11269598
        3.45786302          2.99459722          1.22253920
        0.00000000          2.99459722          3.66761759
```

We can visualize the duplicated structure with VESTA as the following:



8.5 Creation of monolayers and heterostructured slabs

Many emerging 2D materials are being simulated using DFT and other codes to see what their approximate properties will be in many situations that may be difficult to replicate experimentally. Creating monolayers of crystals for calculations, especially in an arbitrary orientation, can sound daunting but this section is intended to dispel that notion and make the construction of arbitrary numbers of atomic layers in oriented crystals and heterostructures simplified.

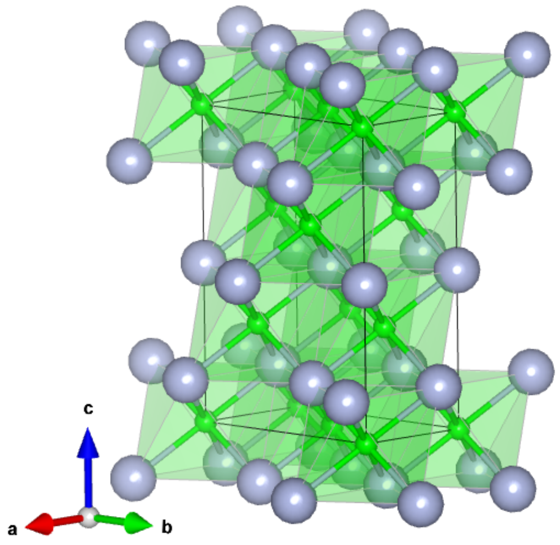
For the first example in this section, we will use Atomsk to create a (111)-oriented ZrN crystal that we will later modify to be tri- bi- and monolayers (without any substrate crystals or layers) of ZrN. We can begin with some simple Atomsk commands like those that we have discussed before:

```
uname@ubuntu:~$ atomsk --create rocksalt 4.5675 Zr N orient [1-10] [11-2] [111] ZrN.
    cfg
uname@ubuntu:~$ atomsk ZrN.cfg vasp
```

The resulting POSCAR file will look like the following:

```
# Rocksalt ZrN oriented X=[1-10] Y=[11-2] Z=[111].
1.000000
       3.22971022       0.00000000       0.00000000
       0.00000000       5.59402220       0.00000000
       0.00000000       0.00000000       7.91114206
 Zr  N
     6       6
Cartesian
       0.00000000       0.00000000       0.00000000
       0.00000000       3.72934815       5.27409473
       0.00000000       1.86467405       2.63704733
       1.61485511       4.66168515       2.63704733
       1.61485511       0.93233705       5.27409473
       1.61485511       2.79701110       0.00000000
       1.61485511       0.93233705       1.31852370
       1.61485511       4.66168515       6.59261836
       1.61485511       2.79701110       3.95557103
       0.00000000       1.86467405       6.59261836
       0.00000000       3.72934815       1.31852370
       0.00000000       0.00000000       3.95557103
```

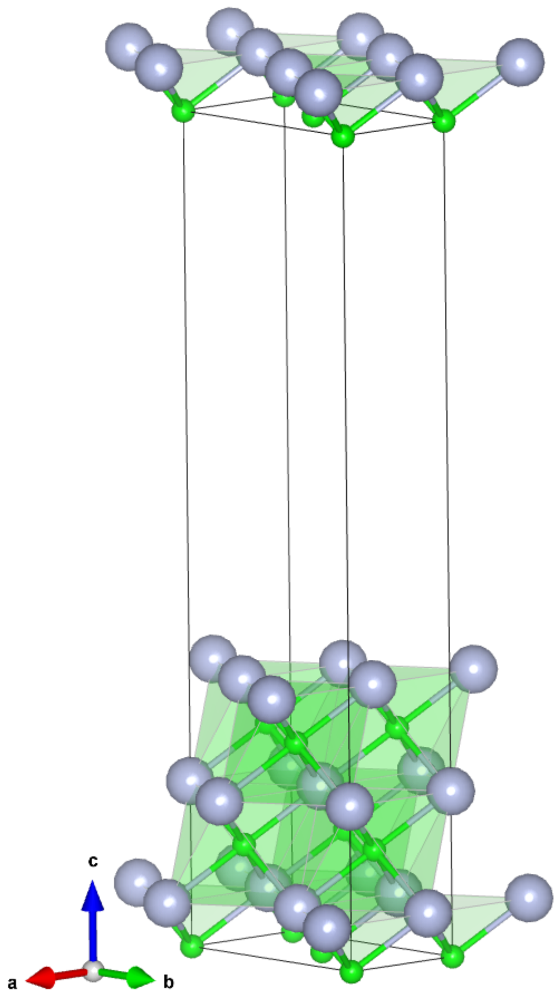
And can be visualized with VESTA:



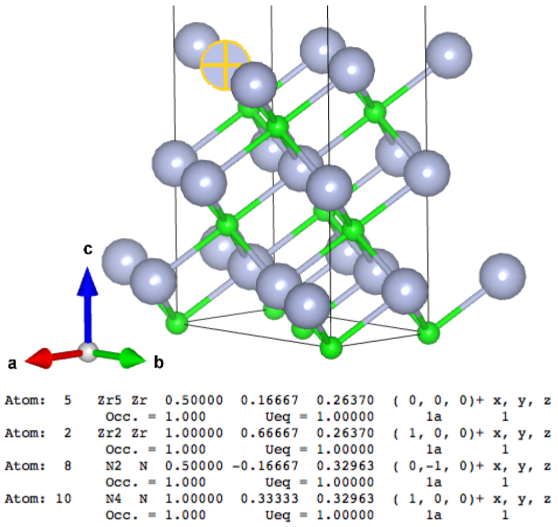
As it stands, this is a bulk crystal of ZrN with its c-axis aligned with the z-axis of the space. We can however turn this into a trilayer of ZrN very rapidly by just adding length to the z-lattice vector in the POSCAR file. This is just done manually and a length that is 'large enough' so that there will be minimal or almost no interactions of atoms at the bottom of the unit cell (because of periodicity) with the atoms on the topmost layer of atoms. In this case I just set the z-component to 20:

```
# Rocksalt ZrN oriented X=[1-10] Y=[11-2] Z=[111].
1.000000
       3.22971022          0.00000000          0.00000000
       0.00000000          5.59402220          0.00000000
       0.00000000          0.00000000         20.00000000
 Zr   N
     6         6
Cartesian
       0.00000000          0.00000000          0.00000000
       0.00000000          3.72934815          5.27409473
       0.00000000          1.86467405          2.63704733
       1.61485511          4.66168515          2.63704733
       1.61485511          0.93233705          5.27409473
       1.61485511          2.79701110          0.00000000
       1.61485511          0.93233705          1.31852370
       1.61485511          4.66168515          6.59261836
       1.61485511          2.79701110          3.95557103
       0.00000000          1.86467405          6.59261836
       0.00000000          3.72934815          1.31852370
       0.00000000          0.00000000          3.95557103
```

Like always, we can visualize the crystal using VESTA:



Now, just by inspection, we can see that there are three ZrN layers (note that this is distinct from atomic layers because of the rocksalt structure and chemistry). Using the VESTA selection tool, we can determine which atoms are in the top ZrN layer: these turn out to be atoms 5, 2, 8, and 10 in the POSCAR file (please see the below two images showing how this is done in VESTA as well as the VESTA terminal output returning the atom numbers):

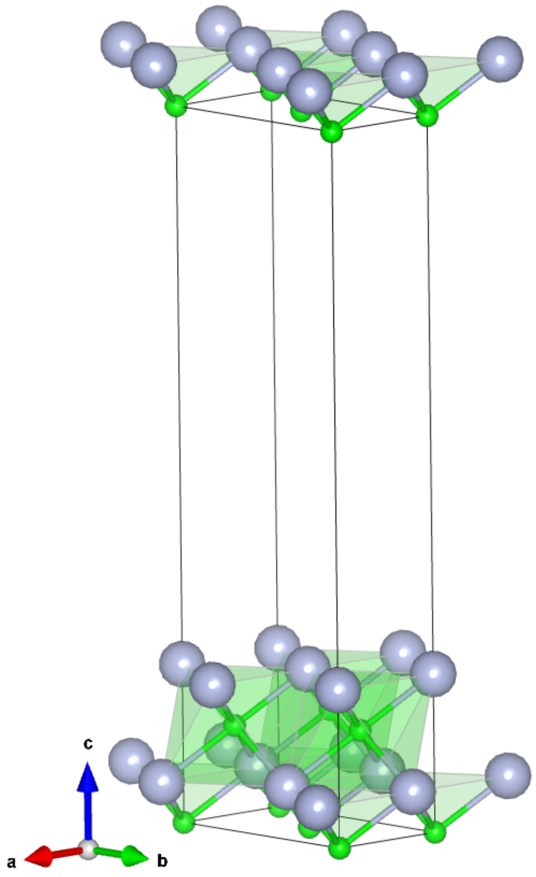


```
Atom:  5   Zr5 Zr  0.50000  0.16667  0.26370  ( 0, 0, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
Atom:  2   Zr2 Zr  1.00000  0.66667  0.26370  ( 1, 0, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
Atom:  8    N2  N  0.50000 -0.16667  0.32963  ( 0,-1, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
Atom: 10    N4  N  1.00000  0.33333  0.32963  ( 1, 0, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
```

Then, after deleting atoms 5, 2, 8, and 10 from the POSCAR file, we will be left with the following POSCAR file:

```
# Rocksalt ZrN oriented X=[1-10] Y=[11-2] Z=[111].
1.000000
       3.22971022          0.00000000          0.00000000
       0.00000000          5.59402220          0.00000000
       0.00000000          0.00000000         20.00000000
 Zr   N
      4         4
selective dynamics
Cartesian
       0.00000000          0.00000000          0.00000000
       0.00000000          1.86467405          2.63704733
       1.61485511          4.66168515          2.63704733
       1.61485511          2.79701110          0.00000000
       1.61485511          0.93233705          1.31852370
       1.61485511          2.79701110          3.95557103
       0.00000000          3.72934815          1.31852370
       0.00000000          0.00000000          3.95557103
```

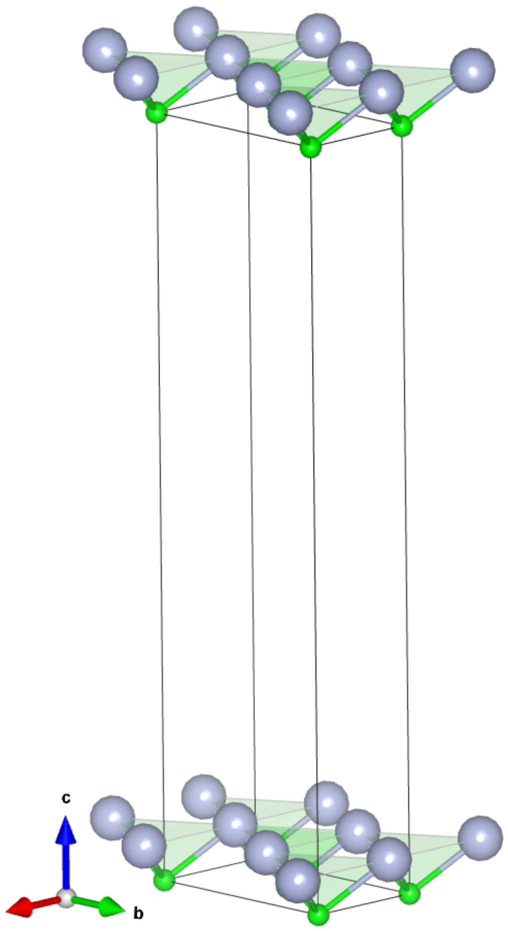
Which can be visualized with VESTA:



In the same way, we can create a monolayer by selectively deleting the unwanted top layer of atoms (in this case atoms 8, 6, 3, and 2 in the POSCAR file as determined by VESTA). The resulting POSCAR file will look like the following (note the T T T characters, which are known to VASP as selective dynamics, following the atomic positions, this is the VASP means of saying that, for specific types of calculations, those atoms are able to move in the x-, y-, and z-coordinates):

```
# Rocksalt ZrN oriented X=[1-10] Y=[11-2] Z=[111].
1.000000
       3.22971022          0.00000000          0.00000000
       0.00000000          5.59402220          0.00000000
       0.00000000          0.00000000         20.00000000
 Zr   N
      2         2
selective dynamics
Cartesian
       0.00000000          0.00000000          0.00000000    T   T   T
       1.61485511          2.79701110          0.00000000    T   T   T
       1.61485511          0.93233705          1.31852370    T   T   T
       0.00000000          3.72934815          1.31852370    T   T   T
```

The monolayer of ZrN can be visualized with VESTA:



Atomsk has quite a few useful tools, not the least of which is the merge tool. As its name may imply, the merge tool allows a user to merge two or more crystal files along a certain direction and in any ordering that the user desires. The Atomsk option is –merge and is usually done after using the –create option to make at least two crystals. In practice it is most simple to create the initial crystal files in the .cfg format, retain that format during the merge operation, and then convert to your desired format at the end of the operations.

In the following example using the Atomsk code, we generate a crystallographic input file to calculate the relaxation of a (111)-oriented monolayer of TiN as a slab on an AlN substrate that is treated as immovable later on with VASP. A merged TiN/AlN slab was created in Atomsk with the following commands:

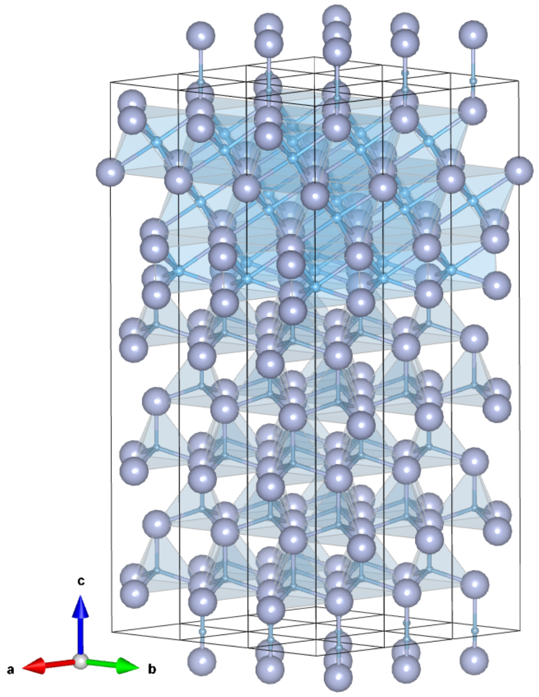
```
uname@ubuntu:~$ atomsk --create wurtzite 3.1117 4.9778 Al N AlN.cfg
uname@ubuntu:~$ atomsk --create rocksalt 4.235 Ti N orient [1-10] [11-2] [111] TiN.
    cfg
uname@ubuntu:~$ atomsk --merge Z 2 AlN.cfg TiN.cfg TiN-111_on_AlN.cfg
```

```
uname@ubuntu:~$ atomsk TiN-111_on_AlN.cfg vasp
```

The resulting POSCAR file will look like the following:

```
# AlN with wurtzite structure with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
1.000000
       3.11170000          0.00000000          0.00000000
      -1.55585000          2.69481125          0.00000000
       0.00000000          0.00000000         22.26863517
 Al   N   Ti
 6    9   3
Cartesian
      -0.00000002          1.79654084          0.00000000
       1.55585002          0.89827041          2.48890011
      -0.00000002          1.79654084          4.97779999
       1.55585002          0.89827041          7.46670009
      -0.00000002          1.79654084          9.95559998
       1.55585002          0.89827041         12.44449986
      -0.00000002          1.79654084          1.86667502
       1.55585002          0.89827041          4.35557513
      -0.00000002          1.79654084          6.84447501
       1.55585002          0.89827041          9.33337490
      -0.00000002          1.79654084         11.82227500
       1.55585002          0.89827041         14.31117488
       1.49729860          2.59339726         18.60101768
       0.00000000          1.72893150         21.04609601
       0.00000002          3.45786304         16.15593913
       1.49729860          4.32232876         17.37847829
       1.49729862          0.86446578         19.82355684
       1.49729860          2.59339726         14.93339997
```

And the crystal itself can be visualized with a program like VESTA:

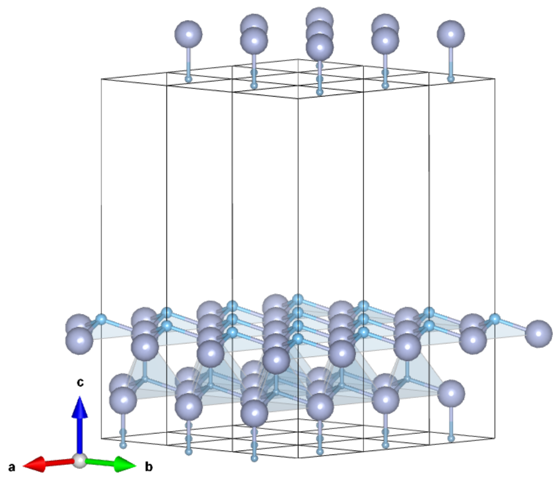


This may be an interesting place to stop if your desire is to make extremely tightly packed quantum well layers. We can also take this a step further to making a monolayer of (111)-oriented TiN on top of a AlN slab that has n nanometers of vacuum above the TiN surface. This can be done by directly manipulating the POSCAR file that we just generated to the point that it becomes the following:

```
# AlN with wurtzite structure with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
1.000000
      3.11170000        0.00000000        0.00000000
     -1.55585000        2.69481125        0.00000000
      0.00000000        0.00000000       15.00000000
 Al   N   Ti
 2    2   1
Cartesian
     -0.00000002        1.79654084        0.00000000
      1.55585002        0.89827041        2.48940005
     -0.00000002        1.79654084        1.86704995
      1.55585002        0.89827041        4.35645000
      0.00000000        0.00000000        4.97879999
```

We have done several things here, all of which were performed by hand and require no complicated figuring. First, we manipulated the c-axis lattice vector to be 15 as that will be 'large enough' for a quantity of vacuum above the surface. Next, we searched for what atoms are where in VESTA and deleted them from the POSCAR file coordinates list, making sure to not forget what species appear where in the list of atomic coordinates. Finally, we changed the numbers following Al, N, and Ti to represent the new quantities of atoms in the crystal after our manipulation.

The (111)-oriented TiN layer on top of the AlN crystal can be visualized using VESTA:



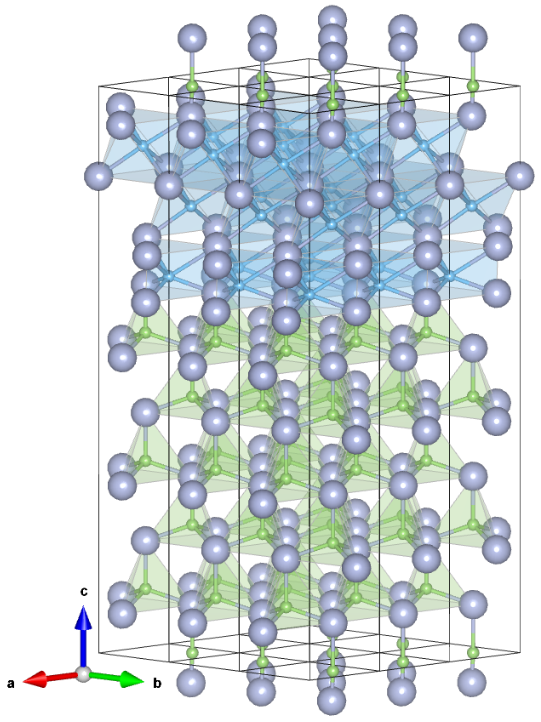
The same operations can be performed on a TiN/GaN heterostructure using Atomsk commands and manual manipulation of POSCAR files with the following commands:

```
uname@ubuntu:~$ atomsk --create wurtzite 3.18 5.166 Ga N -duplicate 1 1 2
    GaN_bilayer.cfg
uname@ubuntu:~$ atomsk --create rocksalt 4.235 Ti N orient [1-10] [11-2] [111] TiN.
    cfg # this file needs to be modified to remove the top two layers of atoms so
    convert to vasp POSCAR, edit, then back to .cfg
uname@ubuntu:~$ atomsk --merge Z 2 GaN_bilayer.cfg TiN_monolayer.cfg TiN_GaN.cfg
uname@ubuntu:~$ atomsk TiN_GaN.cfg vasp
```

The resulting VASP POSCAR file of the unmodified TiN/GaN crystal heterostructure will look like the following:

```
# GaN with wurtzite structure with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
1.000000
       3.18000000         0.00000000         0.00000000
      -1.59000000         2.75396078         0.00000000
       0.00000000         0.00000000        22.83323517
 Ga   N   Ti
      6        9        3
Cartesian
      -0.00000002         1.83597386         0.00000000
       1.59000002         0.91798692         2.58300014
      -0.00000002         1.83597386         5.16600005
       1.59000002         0.91798692         7.74899996
      -0.00000002         1.83597386        10.33200010
       1.59000002         0.91798692        12.91500001
      -0.00000002         1.83597386         1.93724999
       1.59000002         0.91798692         4.52025013
      -0.00000002         1.83597386         7.10325004
       1.59000002         0.91798692         9.68624995
      -0.00000002         1.83597386        12.26925009
       1.59000002         0.91798692        14.85225000
       1.49729860         0.86446578        16.72053928
       1.49729861         2.59339727        19.16561754
       0.00000002         1.72893148        21.61069604
      -0.00000002         3.45786305        20.38815691
      1.49729863          4.32232875         17.94307841
      1.49729861          2.59339727         15.49799992
```

And can be visualized with VESTA (where the green atoms are Ga, the blue are Ti, and the grey are N):



We can add selective dynamics and a vacuum above the crystal so that all of the Ti atoms are movable, the N atoms bonded to the Ti atoms are movable, and the top layer of the GaN crystal is also movable. Please see the following modified POSCAR file:

```
# GaN with wurtzite structure with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001].
1.000000
      3.18000000          0.00000000          0.00000000
     -1.59000000          2.75396078          0.00000000
      0.00000000          0.00000000         40.00000000
 Ga   N   Ti
     6        9         3
Cartesian
     -0.00000002          1.83597386          0.00000000   T   T   T
      1.59000002          0.91798692          2.58300014   F   F   F
     -0.00000002          1.83597386          5.16600005   F   F   F
      1.59000002          0.91798692          7.74899996   F   F   F
     -0.00000002          1.83597386         10.33200010   F   F   F
      1.59000002          0.91798692         12.91500001   T   T   T
     -0.00000002          1.83597386          1.93724999   T   T   T
      1.59000002          0.91798692          4.52025013   F   F   F
     -0.00000002          1.83597386          7.10325004   F   F   F
```

```
      1.59000002          0.91798692          9.68624995   F   F   F
     -0.00000002          1.83597386         12.26925009   T   T   T
      1.59000002          0.91798692         14.85225000   F   F   F
      1.49729860          0.86446578         16.72053928   F   F   F
      1.49729861          2.59339727         19.16561754   F   F   F
      0.00000002          1.72893148         21.61069604   T   T   T
     -0.00000002          3.45786305         20.38815691   T   T   T
      1.49729863          4.32232875         17.94307841   T   T   T
      1.49729861          2.59339727         15.49799992   T   T   T
```

We expand upon the selective dynamics from before by adding the familiar T T T as well as F F F for all atoms that we want to treat as immovable.

8.6 Adding randomness to a crystal

For several reasons like simulating disorder and its associated properties, and also in the creation of scientific visualizations that convey disorder within a crystal, we sometimes desire to add randomness to a crystal structure. Atomsk can handle this very quickly for all sorts of file formats.

As an example of this, I have selected Sb_2S_3 as an interesting example partially because it is a material of interest to my own research, and partially because it has a relatively large unit cell where we can readily visualize the impact of randomness and disorder. This crystal information was collected from the Materials Project and is cataloged under mp-998972. The Materials Project is a fantastic resource that I use very frequently for my own research. The file I collected is in the .cif format and is the conventional standard given by the Materials Project.

Please see the following example of adding randomness to the structure of Sb_2S_3 with Atomsk:

The first step is to convert the .cif file to a VASP POSCAR file so that viewing what specific changes were made to the structure are more simple. This can be done with the following:

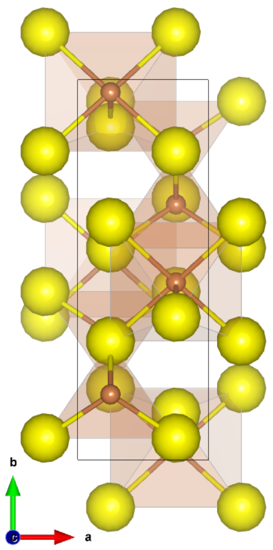
```
uname@ubuntu:~/Desktop$ atomsk Sb2S3_mp-2809_conventional_standard.cif vasp
  ___________________________________________________________
 |              ___________                                  |
 |     o---o    A T O M S K                                  |
 |   o---o|     Version Beta-0.11.1                          |
 |   |   |o     (C) 2010 Pierre Hirel                        |
 |   o---o      https://atomsk.univ-lille.fr                 |
 |___________________________________________________________|
 *** Working out of office hours? You should sleep sometimes. :-)
 >>> Opening the input file: Sb2S3_mp-2809_conventional_standard.cif
 ..> Input file was read successfully (20 atoms).
 >>> Writing output file(s) (20 atoms):
 ..> Successfully wrote POSCAR file: POSCAR
 \o/ Program terminated successfully!
     Total time: 0.008 s.; CPU time: 0.010 s.
```

This will create the following VASP POSCAR file for the Sb_2S_3 structure, note the atomic positions in the undisturbed state:

```
# Sb8 S12
1.000000
      3.87034000          0.00000000          0.00000000
      0.00000000         11.23080100          0.00000000
      0.00000000          0.00000000         12.13389700
 Sb   S
     8       12
Cartesian
```

```
          2.90275500          5.24024682         10.42367275
          0.96758500          5.99055418          1.71022425
          0.96758500         10.85564732          7.77717275
          2.90275500          0.37515368          4.35672425
          0.96758500          1.92956392         11.72180559
          2.90275500          9.30123708          0.41209141
          2.90275500          7.54496442          6.47903991
          0.96758500          3.68583658          5.65485709
          2.90275500          0.64200874         10.62803185
          0.96758500         10.58879226          1.50586515
          0.96758500          6.25740924          7.57281365
          2.90275500          4.97339176          4.56108335
          2.90275500          4.21275207          0.54955633
          0.96758500          7.01804893         11.58434067
          0.96758500          9.82815257          5.51739217
          2.90275500          1.40264843          6.61650483
          0.96758500          2.11071674          3.64987622
          2.90275500          9.12008426          8.48402078
          2.90275500          7.72611724          2.41707228
          0.96758500          3.50468376          9.71682472
```

Visualizing the crystal at this stage will result in the following perfect crystal. The polyhedra are a good guide to the local symmetry. S atoms are yellow, Sb atoms are in brown in this figure:



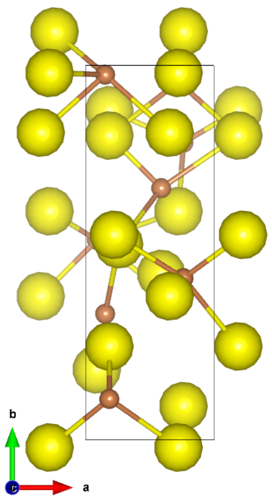
Next, we can use the -disturb option with the value 1.0 to add a random displacement d in the x, y, and z directions of a pseudo-random value in the range $0\text{Å} \leq d \leq 1\text{Å}$:

```
uname@ubuntu:~/Desktop$ atomsk Sb2S3_mp-2809_conventional_standard.cif -disturb 1.0
    vasp
  ___________________________________________________________
 |                    ___________                            |
 |      o---o         A T O M S K                            |
 |    o---o|          Version Beta-0.11.1                    |
 |    |   |o          (C) 2010 Pierre Hirel                  |
 |     o——o      https://atomsk.univ-lille.fr          |
 |_____________________________________________________|
 *** Working out of office hours? You should sleep sometimes. :-)
 >>> Opening the input file: Sb2S3_mp-2809_conventional_standard.cif
 ..> Input file was read successfully (20 atoms).
 >>> Applying a perturbation to atom positions,
 ..> maximum magnitude: 1.000 A.
 ..> Atom positions were disturbed.
 >>> Writing output file(s) (20 atoms):
 ..> Successfully wrote POSCAR file: POSCAR
 \o/ Program terminated successfully!
     Total time: 0.009 s.; CPU time: 0.010 s.
```

Which will modify the previously given VASP POSCAR file for Sb_2S_3 to become the following partially-randomized structure (this time, note that the atomic positions have changed slightly compared to the previous):

```
# Sb8 S12
1.000000
      3.87034000        0.00000000        0.00000000
      0.00000000       11.23080100        0.00000000
      0.00000000        0.00000000       12.13389700
 Sb   S
     8       12
Cartesian
      2.93948807        4.85011933       10.84293948
      0.33547413        5.99492757        1.11527381
      0.58839192       10.95104558        8.51770903
      2.78837768       -0.49382162        4.41095205
      0.71348945        1.22749775       11.73497340
      2.93948807        8.91110958        0.83135814
      2.27064413        7.54933781        5.88408947
      0.58839192        3.78123484        6.39539338
      2.78837768       -0.22696656       10.68225964
      0.71348945        9.88672609        1.51903296
      1.00431807        5.86728174        7.99208038
      2.27064413        4.97776516        3.96613291
      2.52356192        4.30815033        1.29009262
      0.85320768        6.14907363       11.63856847
      0.71348945        9.12608640        5.53055998
      2.93948807        1.01252093        7.03577156
      0.33547413        2.11509013        3.05492578
      2.52356192        9.21548252        9.22455707
      2.78837768        6.85714195        2.47130008
      0.71348945        2.80261759        9.72999252
```

Visualizing the crystal at this stage will result in the following defected, pseudo-randomized crystal, note the major differences between this and the perfect crystal given earlier. S atoms are yellow, Sb atoms are in brown in this figure:


b
a

9 Shell Scripts for DFT Calculations with VASP

VASP (the The Vienna Ab initio Simulation Package) [12, 13, 14] is a very popular closed-source, commercially-available code that will perform DFT, molecular dynamics and much more. VASP became a useful program for my own research but I found that attempting to manipulate the many input files all manually was tedious and way beyond the scope of what I wanted to do. But hooray! We have BASH and the shell to do most of the work for us!

VASP requires several input files in order to run including the POSCAR file which stores the atomic positions and species with their lattice vectors and information about selective dynamics, the POTCAR file which stores the pseudopotentials for the specific calculation, the INCAR file which stores the specific directions that VASP will follow during the calculation(s), and the KPOINTS file which stores information on how the space should be divided for the calculation.

It would be tedious to attempt creating these files all manually every time that you want to run VASP, especially considering that frequently you want to perform many calculations with slightly tweaked parameters. So, to circumvent this, we can create scripts that generate all (or at least most) of the input files for us every time we want to run a calculation or series of calculations. This can be especially handy when trying to mathematically manipulate the POSCAR file for every point in a calculation, or when you are trying to create POTCAR files which are a chore to make manually.

In the following sections, we will discuss shell scripting to automate VASP and make our lives a little easier when running first principles calculations. We might even have some fun along the way!

9.1 Compiling VASP 5.3

VASP is fairly straightforward to compile and has an easy to use build script associated with it. Similar to compilation of the Exciting code covered in section 7.1 or the compilation of Quantum ESPRESSO described in section 6.1, the main part of the heavy lifting is handled by the code and the make files. The main burden on the user is to make sure that the appropriate packages are available for VASP (or whatever code you are using for that matter) in the normal places that the code will search for them.

After successfully running the build script, something to the effect of the following should be printed to the terminal:

```
   Elapsed time = 6m50s
Successfully built the vasp
-rwxrwx--- 1 uname uname 117266472 Oct 21 06:40 vasp
-rwxrwx--- 1 uname uname 117266472 Oct 21 06:40 vasp.kpt
-rw-rw---- 1 uname uname     15381 Oct 21 06:33 vaspxml.mod
```

And, navigating into the executable directory, we can check to see that the file which has been compiled is the executable that we want:

```
uname@uname:~/vasp> cd vasp.5.3/
uname@uname:~/vasp/vasp.5.3> ls vasp
vasp
uname@uname:~/vasp/vasp.5.3> file vasp
vasp: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
    interpreter /lib64/l, BuildID[sha1]=ce6cf06f4d54cde6e6d6605834jh4d534a2029e6,
    for GNU/Linux 3.2.0, with debug_info, not stripped
```

```
uname@uname:~/vasp/vasp.5.3> ./vasp
[Thu Oct 21 08:05:02 2021] [unknown] Fatal error in MPI_Init: Other MPI error, error
    stack:
MPIR_Init_thread(537):
MPID_Init(246).......: channel initialization failed
MPID_Init(647).......:  PMI2 init failed: 1
forrtl: error (76): Abort trap signal
Image              PC                Routine            Line     Source
vasp               0000000026650FA4  for__signal_handl  Unknown  Unknown
libpthread-2.26.s  00002AAAAC4E92D0  Unknown            Unknown  Unknown
libc-2.26.so       00002AAAACA66420  gsignal            Unknown  Unknown
libc-2.26.so       00002AAAACA67A01  abort              Unknown  Unknown
libmpich_intel.so  00002AAAABB77228  Unknown            Unknown  Unknown
libmpich_intel.so  00002AAAABB00062  MPIR_Handle_fatal  Unknown  Unknown
libmpich_intel.so  00002AAAABB00156  MPIR_Err_return_c  Unknown  Unknown
libmpich_intel.so  00002AAAABA84994  MPI_Init           Unknown  Unknown
libdarshan.so      00002AAAAB524711  PMPI_Init          Unknown  Unknown
libmpich_intel.so  00002AAAABAD1D67  MPI_INIT           Unknown  Unknown
vasp               000000002006481D  Unknown            Unknown  Unknown
vasp               000000002007CCD0  Unknown            Unknown  Unknown
vasp               000000002000E38D  Unknown            Unknown  Unknown
vasp               000000002000DB12  Unknown            Unknown  Unknown
libc-2.26.so       00002AAAACA5134A  __libc_start_main  Unknown  Unknown
vasp               000000002000DA2A  Unknown            Unknown  Unknown
Aborted
```

9.2 Some common error sources in VASP

As is common with me, I encounter all sorts of errors when I run calculations, especially when I am trying a new-to-me code. Here is a partial list of some error sources and potential solutions that I have found for use with VASP 5. I am including these items with the presentation first of the error you may see, and then a way you may attempt to resolve the error.

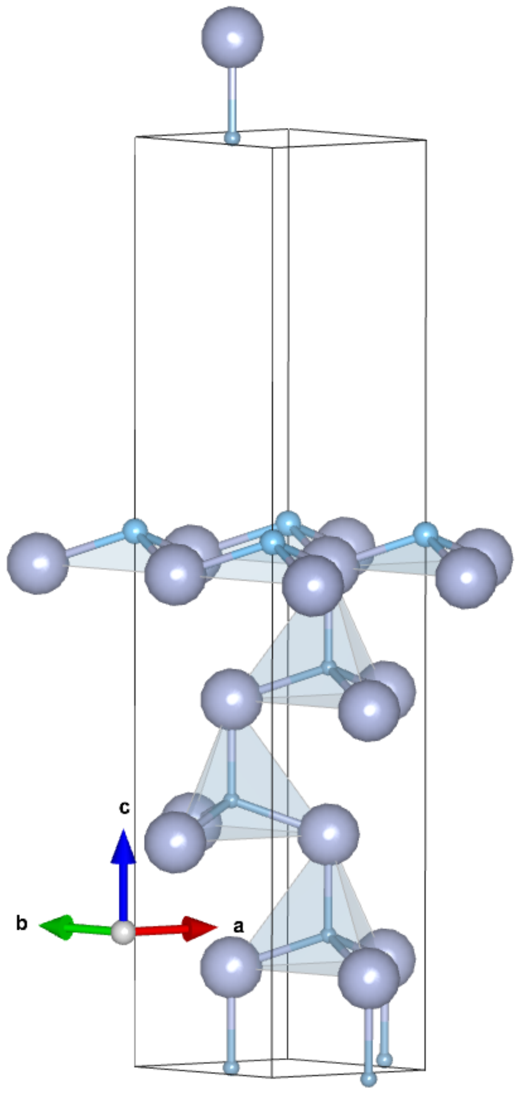
Error	Potential Resolution
Error: LAPACK: Routine ZPOTRF failed! 1 1 1	Several things could cause this issue, I've found it can be resolved most reliably by including ALGO = Fast or ALGO = Normal in your INCAR file or by setting a constant value for POTIM in your INCAR file and increasing that value in steps of 20% from until you calculation runs properly. Alternatively, if nothing seems to be working, you can also try to disable use of LAPACK in your calculation(s)
Error: You find in your log file many errors with file locations pointing to your POSCAR file	Make sure that you have a valid POSCAR file, especially check to make sure that the file's header is readable by VASP
Error: VASP reports in the log file something like the following: "Your highest band is occupied at some k-points! ... Please increase the parameter NBANDS in file INCAR ..."	By default the value of NBANDS in your INCAR file should be either the sum of half the number of valence electrons and half the number of ions in the calculation or 60% of the number of valence electrons in the calculation, whichever value is larger. Try increasing the value of NBANDS manually in your INCAR file by making these calculations for yourself and adding several percent to the number of bands until the calculation runs without the error. Additionally, the value of NBANDS should be divisible by the number of cores (per node) that you're using to run the calculation

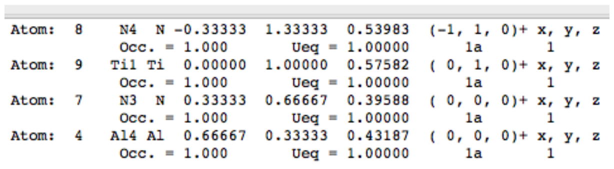
9.3 Relaxation of a (111)-oriented TiN monolayer on AlN

Of interest to some of my research is the relaxation of thin crystalline films on top of distinct host substrate crystals because the crystal structure of the substrate will influence the crystal structure of the thin film that was deposited onto the substrate and therefore influence the film's properties in a variety of ways. VASP has the capability to calculate the relaxed (geometrically optimized to have equilibrium energy) structure of crystal heterostructures in this way. A convenient means of creating this crystal structure for computational simulation is with Atomsk (see section 8) using the following commands:

```
atomsk --create wurtzite 3.1117 4.9778 Al N AlN.cfg
atomsk --create rocksalt 4.235 Ti N orient [1-10] [11-2] [111] TiN.cfg
atomsk --merge Z 2 AlN.cfg TiN.cfg TiN-111_on_AlN.cfg
atomsk TiN-111_on_AlN.cfg vasp
```

Our calculation will focus on the following structure that was created using Atomsk (for more details on Atomsk, please see section 8) (the following text has more enumeration than is normal with specifically added labeling for clarity; it is a terminal output from VESTA [15] reading the POSCAR file that Atomsk has created, and all of the text following the z-coordinate needs to be removed if intended to be used in a calculation). I have labeled which atoms are where in the calculation for ease of later reference when we considered which atoms will be able to move during the structural relaxation and which will not. For the time being, we will want to see the relaxation of the top Ti and N atoms, as well as the first layer of Al and N atoms. These were determined by using the atom selection function in VESTA. The structure as well as the selected atoms for determining which atoms we want to be movable in the relaxation are seen below as a screenshot from the VESTA software (Ti atoms are light blue, N atoms are gray, Al atoms are greenish-blue):





```
Atom:  8    N4  N -0.33333  1.33333  0.53983  (-1, 1, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
Atom:  9   Ti1 Ti  0.00000  1.00000  0.57582  ( 0, 1, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
Atom:  7    N3  N  0.33333  0.66667  0.39588  ( 0, 0, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
Atom:  4   Al4 Al  0.66667  0.33333  0.43187  ( 0, 0, 0)+ x, y, z
            Occ. = 1.000        Ueq = 1.00000      1a       1
```

Vesta also reports the structural parameters of this crystal as the following (with my annotations of what atoms we want to be movable, that is):

```
        a          b          c         alpha      beta      gamma
3.11170   3.11170  17.29284   90.0000   90.0000  120.0000

Unit-cell  volume = 145.008110 A^3
```

```

Structure parameters

                          x           y           z
1  Al   Al1          0.33333     0.66667     0.00000
2  Al   Al2          0.66667     0.33333     0.14396
3  Al   Al3          0.33333     0.66667     0.28791
4  Al   Al4          0.66667     0.33333     0.43187 <-- AlN Top Layer
5  N    N1           0.33333     0.66667     0.10797
6  N    N2           0.66667     0.33333     0.25192
7  N    N3           0.33333     0.66667     0.39588 <-- AlN Top Layer
8  N    N4           0.66667     0.33333     0.53983 <-- Top TiN
9  Ti   Ti1          0.00000     0.00000     0.57582 <-- Top TiN
```

A script for running this relaxation as a VASP calculation is included below and written in several sections. The aim of the script is to eliminate much of the annoyance that one might face when attempting to run these calculations on a remote system where you may have to handle lots of scripts and potential files. This script builds all of the POTCAR (potential) files directly and brand new every time that the script is run, as well as the INCAR, KPOINTS, and POSCAR files and a README of what the script is doing and its important parameters.

The first section of this script (since it is intended to be run on a supercomputer system) asks for parameters to create an SBATCH script including things like the calculation time and the executable's location in the system. Next, it asks for the number of elements that you want to use as well as the location of the pseudopotential files that you want to use for the calculations, these will be used near the end to create the POTCAR file based on the atoms that you have selected. After that, we create a README file that stores some important notes about what the calculation was doing, I find these sorts of README files useful when referring back to previous calculations.

The third section creates an SBATCH script for use in queuing on a supercomputer system that has some variables inputting information from what we have defined above. I choose to follow this route because it is tedious to always scroll through a length of text and change parameters that you cherry-pick from different parts of the script. Instead, it is much simpler and leads to fewer forgotten parameter changes when you collate all of your parameters that you frequently change as variables at the outset of your script.

Next, I create a POSCAR file based on the Atomsk commands that I discussed previously. The POSCAR file has an option called 'selective dynamics' which tells VASP that you want to calculate the system allowing only selected atoms to move. Based on the VESTA screenshot of the Atomsk-created crystal structure, we can see that we want atoms 4, 7, 8, and 9 to be movable in the x, y, and z directions. In order to tell this to VASP, we enter T T T for all the atoms that we want to be movable (one 'T' for each direction in x, y, and z), and similarly F F F for all atoms that we want to be immovable in all directions.

Next we use the cat command to create KPOINTS and INCAR files. These can be perused at your own leisure based on the VASP documentation but together they tell VASP what to calculate and how dense of a grid (with respect to the unit cell) to calculate over. A lot of the verbose labeling within the INCAR file comes from various sources on the internet that have shared their INCAR files and I want to thank all of those people very much for their contribution to the concise labeling of what the various INCAR parameters mean and how they can be modified.

Next, we create the POTCAR file based on the atoms that were defined at the top of the script. Here, we use a switch case statement that chooses how to handle the element variables (again residing at the top of the script) based on the Number_of_Elements variable that we define. I use this methodology of my own creation because I find it tedious to make the POTCAR files every time I want to run a new calculation. To create each POTCAR file, the individual elements are concatenated together using the cat command in the terminal. This script just automates that based on parameters that you can set and forget until you change the system that you are calculating. It is important to note here that you need to give the names of the elements in the exact order that they appear in your POSCAR file.

The final portion of this script submits the job to the cluster's queue using the sbatch command on the SBATCH script that we created and then prints to the queue the status of the calculation using the squeue command every 5 seconds until you terminate the script. The purpose of this is to monitor when your calculation begins to run and how long it takes to run on the computer.

It is frequently convenient to run the script with the following string of commands by copying and pasting the script into VIM and then running the script with ./script_name.sh in the normal way:

```
File_Name="TiN_111_monolayer_on_AlN_vasp.sh" ; touch $File_Name ; chmod +x
    $File_Name ; vim $File_Name
```

Please see the following script for running the relaxation of a TiN monolayer on top of an AlN bulk substrate with VASP:

```
#!/bin/bash

Job_Name="TiN_111_monolayer_on_AlN_vasp" # Give the name you want to apply to all
    files here
Job_Time="00:30:00"                      # Give the run time in hh:mm:ss
Module_Name="vasp"                       # Give the name of the module

Module_Location="~/uname/codes/vasp/vasp.5.3/vasp"
# Executable location
Pseudo_Location="~/uname/pseudopotentials/LDA/potpaw_LDA/"
# Location of pseudopotential files for POTCAR, select: LDA, PBE, PW91

Number_of_Elements="3"  # Number of individual elements in the simulation
Element_1="Al"          # Abbreviated name of element 1 in POSCAR
Element_2="N"           # Abbreviated name of element 2 in POSCAR
Element_3="Ti"          # Abbreviated name of element 3 in POSCAR
Element_4="n/a"         # Abbreviated name of element 4 in POSCAR

#File_Name="TiN_111_monolayer_on_AlN_vasp.sh" ; touch $File_Name ; chmod +x
    $File_Name ; vim $File_Name

Date=$(date '+%d/%m/%Y %H:%M:%S')
# Give date in day/month/year hr/min/sec (thanks unix.stackexchange user1293137)

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF

This is a VASP calculation to calculate the relaxation of a (111)-oriented monolayer
    of TiN as a slab on an AlN substrate that is treated as immovable
The TiN/AlN slab was created in Atomsk with the following commands:

atomsk --create wurtzite 3.1117 4.9778 Al N AlN.cfg
atomsk --create rocksalt 4.235 Ti N orient [1-10] [11-2] [111] TiN.cfg
atomsk --merge Z 2 AlN.cfg TiN.cfg TiN-111_on_AlN.cfg
atomsk TiN-111_on_AlN.cfg vasp

The resulting file called POTCAR was modified by hand to add selective dynamics to
    some atoms and remove layers of Ti and N atoms as well
Ordering of the N atoms was also done to make the file readable by VASP

EOF

   echo " Creating file README.txt..."
   echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Begin VASP File Creation :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running script in.${Job_Name}.sh..."
echo " The time is currently $Date "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > ${Job_Name}.sb << EOF
#!/bin/bash
#SBATCH --job-name=$Job_Name
#SBATCH -N 1
#SBATCH -C haswell
#SBATCH -q debug
#SBATCH -t $Job_Time

module load vasp
srun -n32 -c2 --cpu_bind=cores ${Module_Location}

EOF

echo " Creating input file ${Job_Name}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Create POSCAR File for VASP -:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# selective dynamics allows you to fix atom positions with T and F along supercell
    basis vectors
# F and T following 'direct' in POSCAR indicate whether an atom can (T) or cannot (F
    ) move in a direction (x y z)

cat > POSCAR << EOF
# AlN with wurtzite structure with box vectors H1=[2-1-10], H2=[-12-10], H3=[0001]
    and TiN orient x[1-10] y[11-2] z[111].
1.000000
       3.11170000        0.00000000        0.00000000
      -1.55585000        2.69481125        0.00000000
       0.00000000        0.00000000       17.29283517
 Al   N   Ti
     4        4        1
selective dynamics
```

```
Cartesian
     -0.00000002        1.79654084        0.00000000   F  F  F
      1.55585002        0.89827041        2.48939994   F  F  F
     -0.00000002        1.79654084        4.97880006   F  F  F
      1.55585002        0.89827041        7.46820000   T  T  T
     -0.00000002        1.79654084        1.86705000   F  F  F
      1.55585002        0.89827041        4.35644994   F  F  F
     -0.00000002        1.79654084        6.84585006   T  T  T
      1.55585002        0.89827041        9.33525001   T  T  T
      0.00000000        0.00000000        9.95759995   T  T  T
EOF

echo " Creating input file POSCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create KPOINTS File for VASP :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > KPOINTS << EOF
K-Points
0
Monkhorst-Pack
9 9 1
0 0 0
EOF

echo " Creating input file KPOINTS..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create INCAR File for VASP :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > INCAR << EOF
# Full relaxation cell+ion

# General Setup
  System = Sys_Name    # System name for titling calculations
  PREC   = NORMAL      # precision level: NORMAL, MEDIUM, HIGH, LOW
  ENCUT  = 400         # Electron volts for kinetic energy cutoff value
  ISTART = 0           # 0: start a new job, 1: continue a job
  ICHARG = 2           # Charge density from atoms
  ISPIN  = 1           # 1 if calculation is spin polarized, 2 if not

# Electronic Relaxation (SCF)
  NELM    = 60       # Max no. of steps to calculate before halting
  NELMIN  = 2        # Min "                                      "
  NELMDL  = 10       # Number of non-self consistent steps at the beginning
  EDIFF   = 1.0E-05  # Global break condition for the electronic SC-loop
  LREAL   = .FALSE.  # Projection operators evaluated in real-space
  IALGO   = 48       # Electronic algorithm used to optimize the orbitals.
  VOSKOWN = 1        # Determines whether Vosko-Wilk-Nusair interpolation is used
  ADDGRID = .TRUE.   # Improve the grid accuracy

# Ionic Relaxation
  EDIFFG  = -1.0E-04 # Break condition for the ionic relaxation loop.
  NSW     = 25       # Sets the maximum number of ionic steps
  IBRION   = 2          # Relaxation Method: 0-MD 1-qNewton-RaphsonElectronic 2-CG
  ISIF     = 3          # Calculate forces Stress tensor positions cell shape volume
  ADDGRID = .TRUE.      # Additional support grid is used
  SIGMA    = 0.10       # Insulators/semiconductors=0.1  metals=0.05
  ISMEAR   = 0          # Sets partial occupancies fnk for each orbital
                        # -1 Fermi Smear, 0 Gaussian Smear

EOF

echo " Creating input file INCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file POTCAR -:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# This switch/case will automatically generate a POTCAR file based on the options in the header of this file
# Create POTCAR with cat in the order that elements appear in the POSCAR file

case $Number_of_Elements in

  1) echo " One element selected for simulation: $Element_1"
     cat ${Pseudo_Location}${Element_1}/POTCAR > POTCAR ;;
  2) echo " Two elements selected for simulation: $Element_1 $Element_2"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR > POTCAR ;;
  3) echo " Three elements selected for simulation: $Element_1 $Element_2 $Element_3"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR ${Pseudo_Location}${Element_3}/POTCAR > POTCAR ;;
  4) echo " Four elements selected for simulation: $Element_1 $Element_2 $Element_3 $Element_4"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR ${Pseudo_Location}${Element_3}/POTCAR ${Pseudo_Location}${Element_4}/POTCAR > POTCAR ;;

esac

   echo " Creating file POTCAR..."
   echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:- Run VASP Calculation with SBATCH :-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Submitting ${Job_Name}.sb via sbatch..."

sbatch ${Job_Name}.sb

echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:End Elapsed Time Measurement-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

squeue -u uname

```

```
echo " The time is currently $Date "

echo " Success , End of Script"

while [ 1 ] ; do squeue -u uname; sleep 5; done    # continue to update the squeue
    every 5 seconds
```

9.4 Automating many simultaneous calculations with VASP: the O_2 dimer, and data analysis with MATLAB

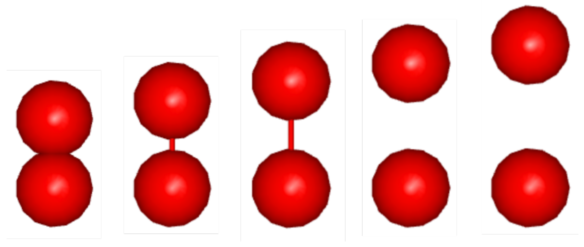
An introductory and useful type of problem to solve with VASP is the ground state energy and equilibrium bond length of a dimer in vacuum. Because it is hard to predict exactly what the equilibrium bond length of a molecule will be, we can 'scan' a space with an arbitrary number of calculations of different atomic configurations and then use some subsequent mathematical analysis to determine the atomic configuration which minimizes the energy of the system. Clever readers may note that this can be done by molecular dynamics but in some cases with larger systems it is more useful to report the energy at every step and run your own fitting of the energy and bulk modulus for example. Here we will discuss a solution to this type of problem for the case of a diatomic molecule using VASP.

Previously, in section 9.3, building and running a single computation in VASP was demonstrated. However, in the case that you might want to run many calculations with slightly varied parameters based on a general framework for a computation, trying to apply the same means of script-making and running individual calculations would be tedious. However, since we are using the shell, we can automate all sorts of things, including the creation and submission of scripts for calculations with many slightly varied parameters to a computing cluster. We will cover means of collecting the data from these multiple calculations in sections 9.8, and 9.9.

Running many computations for slightly tweaked structural parameters is important in the generation of force field files like the EAM and MEAM standards which can be used with programs like LAMMPS. One such code that will accept these series of computations for the generation of force filed files is MEAMfit (discussed further in chapter 10); another such code is potfit (discussed further in chapter 11). In the present section, we will discuss a means of iterating parameters of a system in a shell script in order to create a series of input scripts that may used for running concurrent calculations on similar but distinct systems. This is beneficial for finding many parameters of systems like energy minimization for a given structure, as well as for the previously mentioned generation of force field files.

In this section, we will discuss the generation of a large number of VASP input scripts to calculate the ground state of a diatomic oxygen molecule. It is important to remember that, since oxygen is a ground state triplet, our calculations will need to be spin-polarized (you will need to verify for yourself whether your calculations need to be spin polarized based on the elements and their arrangements you are considering). We will generate a script that is based off of that from section 9.3 but will have different calculation directions for VASP to follow and use a for loop (more information on for loops can be found in section 4.3.1) to generate an arbitrary number of distinct input files based on a range of lattice parameters (which I will use in this section synonymously with the bonding length in the diatomic molecule) that are automatically generated based on inputs of an initial lattice parameter, a number of distinct calculations in the positive and negative bond strain directions, and a maximum positive and negative strain percent for the system. As with other examples in this text, this script is built to be run on a supercomputer system, however there is nothing stopping it from being run on more or less any system with just minor tweaks to the instructions for how to run the program.

Schematically, the general idea of the system that we will be covering is shown below (oxygen atoms and bonds between oxygen atoms are illustrated in red). We will be calculating the ground state energy for a variety of atomic configurations having increasing and decreasing bond length from what is known as the literature value of the diatomic oxygen bond length. Following this illustration is a discussion of how we can implement such a calculation.



The following is a script (divided into several sections to facilitate greater human-readability) which will generate 41 distinct spin-polarized VASP calculations of the the diatomic oxygen system and submit all of them to a supercomputer queue using sbatch commands. The script will generate 1 calculation for the initial guess given for the lattice parameter (e.g., the bonding length), 20 calculations for the case of compressive strain in the bond with 20 steps down to the maximum compressive strain of 2%, and another 20 calculations for the case of tensile strain in the bond with 20 steps up to the maximum tensile strain in the bond. The script will automatically generate a new set of VASP input files and subdirectories for each of the 41 calculations. The number of calculations, maximum strain in the bond(s), and the accuracy thresholds for all of the calculations, the initial guess for lattice parameter or bonding length, and even the individual elements can all be quickly augmented to suit your needs.

In this script's first section, we define a some variables. Much of this will look familiar to the example given in section 9.3. In this case we are only populating a single atomic species and giving the number of elements as 1. We use this section to set several INCAR file parameters including the precision level and the NPAR value (which I have the script calculate automatically for me based on the square root of the number of job nodes that the calculation will use) for parallel computations. We set variables for an initial lattice parameter guess, and the number of calculation steps for the compressive and tensile bond strain cases as well as the maximum strain in each direction. These values are handled automatically by the script using the shell program bc (see section 3.12 for more information on this particular program) to determine a maximum and minimum lattice parameter (or bonding length) for the system as well as a step size between all of the calculation steps. These are important parameters for the for loop that we will set up shortly. The date and the number of jobs are also stored as variables here.

After this, a README file is generated that stores pertinent information about the calculation set and the system that is being simulated. It is also set up so that (in the event that you choose to use the tee command which is covered in section 4.2) the terminal session and all information from the supercomputer queue including calculation times and concurrent jobs will be recorded to the README file.

Next, we begin our main for loop. We use the sequence command here within backticks (please make sure that whatever pdf interpreter you are using here is printing backticks and not apostrophes for the for loop) iterate the remainder of the script for n times in a sequence between the minimum lattice parameter to the maximum lattice parameter (or bonding length) in increments of the step size variable that we had automatically calculated in the beginning of the script. The for loop stores its current value in the variable called lat. For the sake of monitoring the script's progress, I also include a command to print the current lattice parameter and several other reports to the terminal for every step of the for loop.

Inside of this for loop, we begin by creating a syntax for file creation that concatenates the system name with the lattice parameter (stored in the lat variable). At each step of the loop, we create a new directory for the specific lattice parameter that we are on, and navigate into that directory. We then create an SBATCH file with all sorts of variable-stored information from the first part of our script as well as KPOINTS, and INCAR files, and a POTCAR file that is automatically generated based on the number of elements and the specific specie(s) of element that you chose at the beginning of the script.

The main unique feature of the individual calculations handled here is in the POSCAR file where we define a large box as well as the positions of two atoms. The first atom is at the origin and is never modified. The location of the second atom has its z-coordinate defined by the value of lat (the lattice parameter variable that we iterate with the for loop). At every step, a unique POSCAR file is written for the bonding length that we care about at that instant. Finally, we submit the calculation to the queue and then begin on the next step in the loop.

The script concludes by printing the date to the terminal and then updating the terminal with the status of the queue using the squeue command and also printing the date and time. squeue and the date command are run on an infinite loop, updating every five seconds until the user kills the process.

The script can be run by entering the following command into the terminal and then copying and pasting the text of this script into the vim editor that will be opened and passing the command :wq to vim. Upon exiting vim, the script will be executed and calculations will begin to be submitted to the supercomputer queue:

```bash
File_Name="O2_dimer_automated_vasp"; mkdir ${File_Name}; cd ${File_Name}; touch ${
    File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh
    |& tee -a README.txt
```

What follows is the script we have just discussed. Some of the inspiration for this script (in the iteration of lattice parameters directly in the POSCAR file) came in part from the 2017 guides to VASP located at icme.hpc.msstate.edu.

```bash
#!/bin/bash

# Copy and paste the below line into the cluster terminal to make and run the script
     (paste into vim and save with :wq)
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks
     to Byte Commander on Stack Exchange)

# File_Name="O2_dimer_automated_vasp"; mkdir ${File_Name}; cd ${File_Name}; touch ${
    File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh
    |& tee -a README.txt

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Give the Following Variables -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
Job_Name="O2_dimer_automated_vasp" # Give the name you want to apply to all files here

Description="energies of automatically expanded and contracted bond lengths in an O2 dimer, spin-polarized" # Please give a short description of the calculation for the README.txt file
Author="Steven E. Bopp, Materials Science & Engineering"

System_Name="O2_Dimer"  # Give a calculation title for VASP
Number_of_Elements="1"  # Give the number of individual elements in the simulation
Element_1="O"           # Give the symbol of the element 1 in your POSCAR file
Element_2="n/a"         # Give the symbol of the element 2 in your POSCAR file
Element_3="n/a"         # Give the symbol of the element 3 in your POSCAR file
Element_4="n/a"         # Give the symbol of the element 4 in your POSCAR file

Precision_Level="HIGH"  # VASP Precision. Options are: NORMAL, MEDIUM, HIGH, LOW

Job_Time="00:05:00"     # Give the run time in hh:mm:ss
Job_Nodes="1"           # Give the number of nodes to use for the calculation
Job_Queue="debug"       # Give the calculation queue (e.g., 'debug' or 'regular')
Computer_Name="cluster" # Give the name of the cluster

Lattice_Parameter="1.208"
# Give the initial guess interatomic distance between oxygen atoms
Calculation_Steps_in_Each_Direction="20"
# Give the total number of calculation points in each direction (e.g. 20 in the + direction, 20 in the - direction = 41 total including a0)
Lattice_Parameter_Variation="2"
# Give upper and lower bounds of % a0 change e.g., a value of 2 would mean that you want a 2% variation which is a0 +/- 0.02*a0

echo " Running VASP calculations on expanded and contracted lattice parameters of ${System_Name}"
echo " with a0=${Lattice_Parameter}A, a ${Lattice_Parameter_Variation}% maximum a0 variation, and +/- ${Calculation_Steps_in_Each_Direction} individual calculations"

# Give the module name, as well as the executable, and pseudopotential locations
Module_Name="vasp"
Module_Location="~/uname/codes/vasp/vasp.5.3/vasp"
Pseudo_Location="~/uname/pseudopotentials/LDA/potpaw_LDA/"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:- BASH and bc Calculated Variables for VASP Automation :-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Lat_Param_Max=$(echo "scale=2;($Lattice_Parameter)*(    1.0 $Lattice_Parameter_Variation)" | bc)
# Use bc to calculate a0_max
Lat_Param_Min=$(echo "scale=2;($Lattice_Parameter)*(1 - 0.0 $Lattice_Parameter_Variation)" | bc)
# Use bc to calculate a0_min
Step_Size=$(echo "scale=3;( (($Lat_Param_Max) - ($Lat_Param_Min))/($Calculation_Steps_in_Each_Direction*2) )" | bc)
# Use bc to calculate N_steps as step size
# The above calculations determine equal step sizes between a0_min and a0_max, and are explained diagrammatically in the following:
#                  a0_min          a0          a0_max
#                    |<- N_steps ->|<- N_steps ->|
echo " Max Lattice Parameter is ${Lat_Param_Max}, Min Lattice Parameter is ${
    Lat_Param_Min}, Step Size is ${Step_Size}"

Number_of_Jobs=$(echo "scale=2;(($Calculation_Steps_in_Each_Direction*2)+1)" | bc)
Date=$(date '+%d/%m/%Y %H:%M:%S')           # Give date in day/month/year hr/min/
    sec thanks user1293137 from https://unix.stackexchange.com/
NPAR=$(echo "scale=0;sqrt($Job_Nodes)" | bc) # Calculate NPAR to be inserted into
    INCAR file [~sqrt(job cores)]; uses BASH program bc; scale=0 sets bc to round to
     nearest integer (necessary for VASP)

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF
Job Name: ${Job_Name}.sh
This is a ${Module_Name} calculation of the ${System_Name} system to calculate ${
    Description}.
Calculating with ${Number_of_Elements} element(s): ${Element_1}, ${Element_2}, ${
    Element_3}, and ${Element_4} for ${Job_Time} with ${Job_Nodes} job nodes on the
    ${Job_Queue} queue.
Calculated with ${Computer_Name} by ${Author} on $Date.

A transcript of the calculation as seen from the terminal follows:

EOF

echo " Writing file README.txt..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Begin VASP File Creation :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-: Automatically Generate VASP Input Files :-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:- What Files Are Generated is Controlled by the Given Lattice Vars. -:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running script ${Job_Name}.sh..."

echo " The time is currently $Date "

echo " Automatically generating input files for ${Module_Name} "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:- Begin Main for() Loop for VASP File Creation :-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

for lat in `seq -w ${Lat_Param_Min} ${Step_Size} ${Lat_Param_Max}`; do
# ` is a backtick: Everything between backticks is executed by the shell before the
    main command, output is then used by that command

echo " The Lattice Constant Variable (lat) at this step = $lat"

mkdir ${System_Name}_a0_${lat}; cd ${System_Name}_a0_${lat}
# Make and navigate into directory for each iteration in the for() loop

```

```
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > ${Job_Name}_a0_${lat}.sb << EOF
#!/bin/bash
#SBATCH --job-name=${Job_Name}_a0_${lat}
#SBATCH -N ${Job_Nodes}
#SBATCH -C haswell
#SBATCH -q ${Job_Queue}
#SBATCH -t ${Job_Time}

module load ${Module_Name}
srun -n32 -c2 --cpu_bind=cores ${Module_Location}

EOF

echo " Writing input file ${Job_Name}_a0_${lat}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create POSCAR File for VASP -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > POSCAR << !
O atom in a box
1.0
7.0 0.0 0.0
0.0 7.5 0.0
0.0 0.0 8.0
2
cartesian
0 0 0
0 0 $lat
!

echo " Writing input file POSCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create KPOINTS File for VASP :-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > KPOINTS << EOF
Gamma-point only
 0
Monkhorst-Pack
 1  1  1
 0.  0.  0.
EOF

echo " Writing input file KPOINTS..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create INCAR File for VASP :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
cat > INCAR << EOF
# General Setup
  System = ${System_Name}     # Calculation Title
  PREC   = ${Precision_Level} # Options: Normal, Medium, High, Low

  ISMEAR = 0              # Gaussian smearing
  ISPIN  = 2              # Spin Polarize: 1-No 2-Yes
  NSW    = 5              # 5 ionic steps
  IBRION = 2              # use conjugate gradient algorithm

# Parallelization
  NPAR   = ${NPAR}        # approx. SQRT(number of cores)
EOF

echo " Writing input file INCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file POTCAR -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# This switch/case will automatically generate a POTCAR file based on the options in the header of this script
# Create POTCAR with cat in the exact order that elements appear in the POSCAR file

case $Number_of_Elements in

  1) echo " One element selected for simulation: $Element_1"
     cat ${Pseudo_Location}${Element_1}/POTCAR > POTCAR ;;
  2) echo " Two elements selected for simulation: $Element_1 $Element_2"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR > POTCAR ;;
  3) echo " Three elements selected for simulation: $Element_1 $Element_2 $Element_3"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR ${Pseudo_Location}${Element_3}/POTCAR > POTCAR ;;
  4) echo " Four elements selected for simulation: $Element_1 $Element_2 $Element_3 $Element_4"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR ${Pseudo_Location}${Element_3}/POTCAR ${Pseudo_Location}${Element_4}/POTCAR > POTCAR ;;

esac

echo " Writing input file POTCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:- Run VASP Calculation with SBATCH -:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Submitting ${Job_Name}_a0_${lat}.sb via sbatch..."

sbatch ${Job_Name}_a0_${lat}.sb               # Submit job to queue

echo " Running ${Number_of_Jobs} job(s) on the ${Job_Queue} queue with ${Job_Nodes} node(s) per job for ${Job_Time} each"

echo "  Begin:"

#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—: End VASP File Creation —:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—: End Main for() Loop for VASP File Creation —:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—

cd .. # Navigate out of newly created directory

done  # End of main for() loop

#echo " The contents of this directory are now the following:" ; ls # List newly
    created directories

#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:End Elapsed Time Measurement—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—

squeue —u uname

echo " The time is currently $Date "

echo " Dumping terminal session into README.txt"

echo " Success, End of Script, Running squeue on a 5 Second Loop"

while [ 1 ] ; do squeue —u uname; date; sleep 5; done # update the squeue every 5
    seconds
```

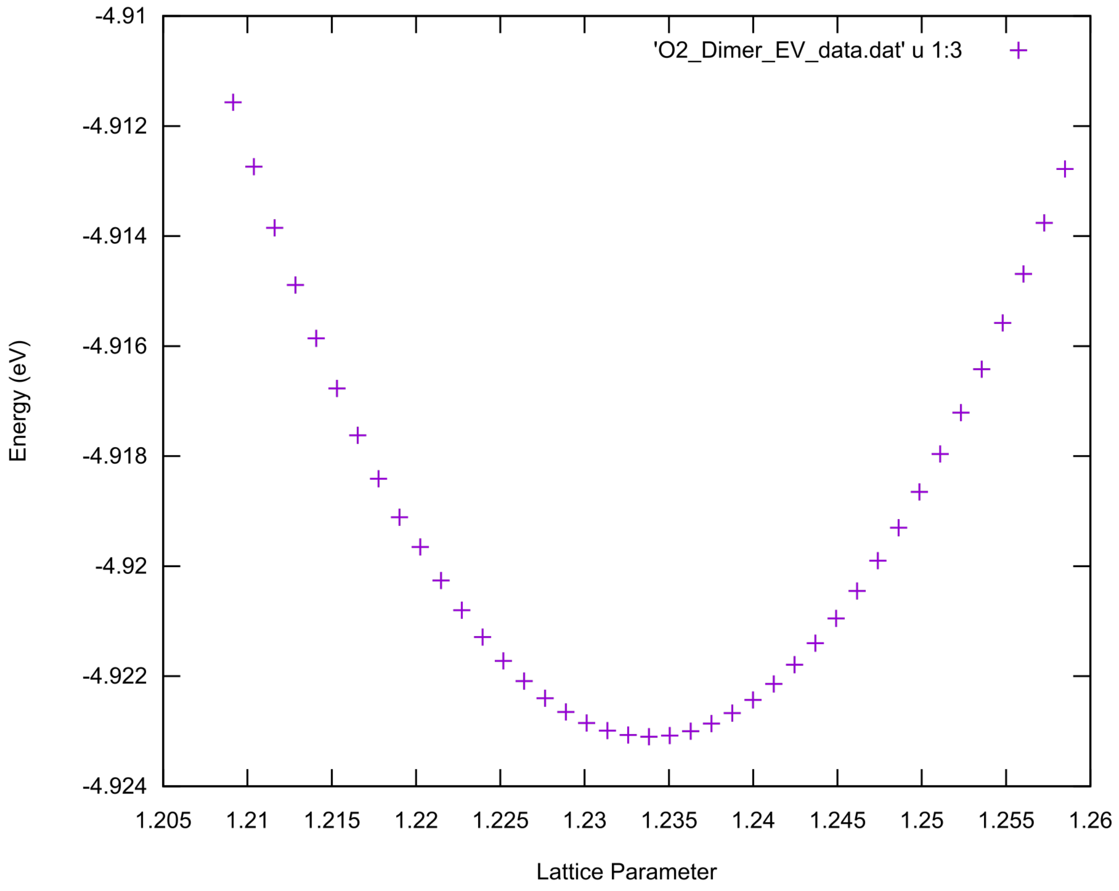
We can see a transcript of the output of this script in the following terminal session which was collected in the README file (as discussed above). For the sake of brevity, some of the README.txt contents have been truncated:

```
uname@uname:~/test_O2_dimer_automated_vasp> head —n 25 README.txt
Job Name: O2_dimer_automated_vasp.sh
This is a vasp calculation of the O2_Dimer system to calculate energies of
    automatically expanded and contracted bond lengths in an O2 dimer, spin—
    polarized.
Calculating with 1 element(s): O, n/a, n/a, and n/a for 00:05:00 with 1 job nodes on
     the debug queue.
Calculated with cluster by Steven E. Bopp, Materials Science & Engineering on
    08/08/2021 00:23:27.

A transcript of the calculation as seen from the terminal follows:

 Writing file README.txt...
  done
 Running script O2_dimer_automated_vasp.sh...
 The time is currently 08/08/2021 00:23:27
 Automatically generating input files for vasp
 The Lattice Constant Variable (lat) at this step = 1.183
 Writing input file O2_dimer_automated_vasp_a0_1.183.sb...
  done
 Writing input file POSCAR...
  done
 Writing input file KPOINTS...
  done
 Writing input file INCAR...
  done
 One element selected for simulation: O
 Writing input file POTCAR...
  done
 Submitting O2_dimer_automated_vasp_a0_1.183.sb via sbatch...
```

And from the center(ish) of the README.txt file, we can see the progression of some jobs on the computer (this is also a nice use of the sed command where you can choose the specific lines in a text file that you would like printed to the terminal):

```
uname@uname:~/test_O2_dimer_automated_vasp> sed -n '110,128p;129q' README.txt
JOBID     NAME          TIME_LIMIT  TIME   SUBMIT_TIME          START_TIME
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:04   2021-08-08T00:23:28  2021-08-08T00:23:29
xxxxxxxx  O2_dimer_aut        5:00  0:04   2021-08-08T00:23:28  2021-08-08T00:23:29
Sun Aug  8 00:23:33 PDT 2021
JOBID     NAME          TIME_LIMIT  TIME   SUBMIT_TIME          START_TIME
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:09   2021-08-08T00:23:28  2021-08-08T00:23:29
xxxxxxxx  O2_dimer_aut        5:00  0:09   2021-08-08T00:23:28  2021-08-08T00:23:29
Sun Aug  8 00:23:39 PDT 2021
JOBID     NAME          TIME_LIMIT  TIME   SUBMIT_TIME          START_TIME
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:00   2021-08-08T00:23:28  N/A
xxxxxxxx  O2_dimer_aut        5:00  0:15   2021-08-08T00:23:28  2021-08-08T00:23:29
xxxxxxxx  O2_dimer_aut        5:00  0:15   2021-08-08T00:23:28  2021-08-08T00:23:29
```

Using Gnuplot (for more information, see section 3.8), we can visualize what the results of a sample calculation for this system would be as follows:



Using some mathematical analysis with MATLAB, we can also determine what the absolute minimum energy value in the calculation set which corresponds to the equilibrium configuration. In section 9.8 we will see how to automatically extract all of the pertinent data from these calculations, suffice it to say here however that we have this data already collected and will import it into MATLAB for analysis. Please see the following .dat file from which the above plot was created:

```
"Lattice Parameter a0" "Total Energy E0" "Energy Per Ion" "Volume Per Ion in A"
1.209143600 −.98231462E+01 −4.91157 256.00
1.210377420 −.98254839E+01 −4.91274 256.00
1.211611240 −.98276915E+01 −4.91385 256.00
1.212845060 −.98297702E+01 −4.91489 256.00
1.214078880 −.98317210E+01 −4.91586 256.00
1.215312700 −.98335454E+01 −4.91677 256.00
1.216546520 −.98352442E+01 −4.91762 256.00
1.217780340 −.98368186E+01 −4.91841 256.00
1.219014160 −.98382295E+01 −4.91911 256.00
1.220247980 −.98393042E+01 −4.91965 256.00
1.221481800 −.98405157E+01 −4.92026 256.00
1.222715620 −.98416073E+01 −4.9208 256.00
1.223949440 −.98425801E+01 −4.92129 256.00
```

```
1.225183260  -.98434348E+01  -4.92172  256.00
1.226417080  -.98441725E+01  -4.92209  256.00
1.227650900  -.98447945E+01  -4.9224  256.00
1.228884720  -.98453017E+01  -4.92265  256.00
1.230118540  -.98456951E+01  -4.92285  256.00
1.231352360  -.98459758E+01  -4.92299  256.00
1.232586180  -.98461446E+01  -4.92307  256.00
1.233820000  -.98462027E+01  -4.9231  256.00
1.235053820  -.98461508E+01  -4.92308  256.00
1.236287640  -.98459901E+01  -4.923  256.00
1.237521460  -.98457215E+01  -4.92286  256.00
1.238755280  -.98453460E+01  -4.92267  256.00
1.239989100  -.98448644E+01  -4.92243  256.00
1.241222920  -.98442777E+01  -4.92214  256.00
1.242456740  -.98435869E+01  -4.92179  256.00
1.243690560  -.98427928E+01  -4.9214  256.00
1.244924380  -.98418963E+01  -4.92095  256.00
1.246158200  -.98408986E+01  -4.92045  256.00
1.247392020  -.98398007E+01  -4.9199  256.00
1.248625840  -.98386035E+01  -4.9193  256.00
1.249859660  -.98373093E+01  -4.91865  256.00
1.251093480  -.98359161E+01  -4.91796  256.00
1.252327300  -.98344279E+01  -4.91721  256.00
1.253561120  -.98328423E+01  -4.91642  256.00
1.254794940  -.98311620E+01  -4.91558  256.00
1.256028760  -.98293878E+01  -4.91469  256.00
1.257262580  -.98275207E+01  -4.91376  256.00
1.258496400  -.98255615E+01  -4.91278  256.00
```

Importing this data (the first and third columns) into MATLAB, we can use splines and the find min functionality to find the minimum energy configuration based on interpolation:

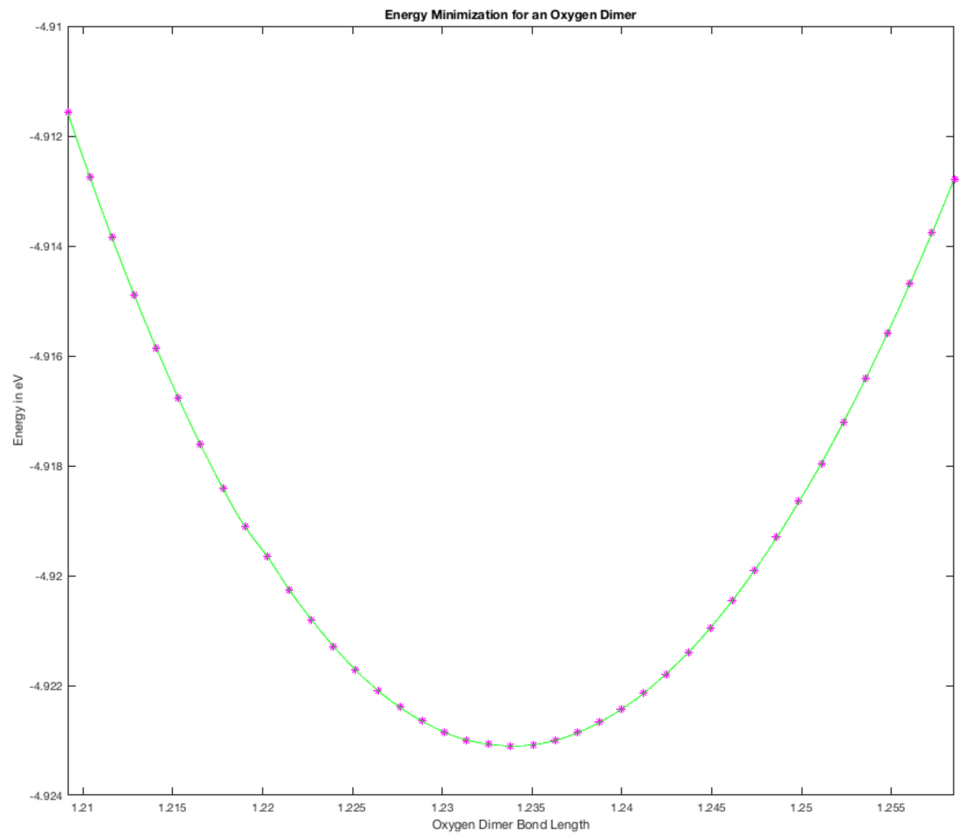
```
%% Oxygen_Dimer_Bond_Length.m
%  Written by Steven E. Bopp

%%
Lattice_Parameter=[1.209143600
1.210377420
1.211611240
1.212845060
1.214078880
1.215312700
1.216546520
1.217780340
1.219014160
1.220247980
1.221481800
1.222715620
1.223949440
1.225183260
1.226417080
1.227650900
1.228884720
1.230118540
1.231352360
1.232586180
1.233820000
1.235053820
1.236287640
```

```
1.237521460
1.238755280
1.239989100
1.241222920
1.242456740
1.243690560
1.244924380
1.246158200
1.247392020
1.248625840
1.249859660
1.251093480
1.252327300
1.253561120
1.254794940
1.256028760
1.257262580
1.258496400];

Energy_Per_Ion=[-4.91157
 -4.91274
 -4.91385
 -4.91489
 -4.91586
 -4.91677
 -4.91762
 -4.91841
 -4.91911
 -4.91965
 -4.92026
 -4.9208
 -4.92129
 -4.92172
 -4.92209
 -4.9224
 -4.92265
 -4.92285
 -4.92299
 -4.92307
 -4.9231
 -4.92308
 -4.923
 -4.92286
 -4.92267
 -4.92243
 -4.92214
 -4.92179
 -4.9214
 -4.92095
 -4.92045
 -4.9199
 -4.9193
 -4.91865
 -4.91796
 -4.91721
 -4.91642
 -4.91558
 -4.91469
 -4.91376
 -4.91278];

%%

E0=Energy_Per_Ion';
x=Lattice_Parameter'; xx=[1.209143600:.0001:1.258496400];
yy=spline(x,E0,xx);         % Create spline interpolate for the lattice constant

figure(1);
plot(x,E0,'m*',xx,yy,'g'); % Plot spline interpolate and Energy with Lattice
    Parameter
xlabel('Oxygen Dimer Bond Length'); ylabel('Energy in eV');
title('Energy Minimization for an Oxygen Dimer');
xlim([1.209143600 1.258496400])
% Pentagram p; hexagram h; diamond d; square s;

indexmin=find(min(yy) == yy);                 % Define indexmin
xmin = xx(indexmin); ymin = yy(indexmin);    % Calculate minimum values
A0 = xmin;
fprintf('Bond Length for an Oxygen Dimer:%g \n',A0) % Display lattice constant and
    error from expected value

```

MATLAB should report the following:

```
Bond Length for an Oxygen Dimer:1.23394
```

MATLAB should also generate the following plot of what it has fit with the spline interpolation:

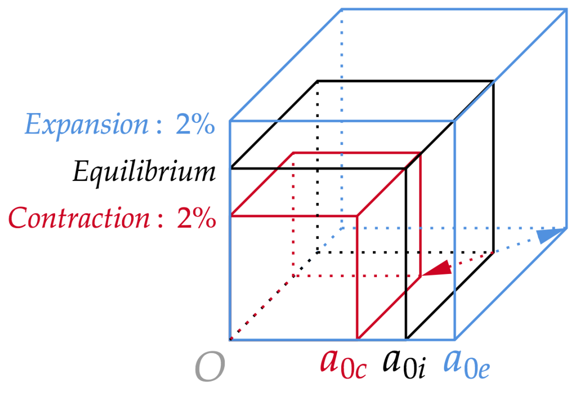


What this specifically means is that, from the spline interpolation, MATLAB has calculated that 1.23394 angstroms should be the equilibrium bond length for an oxygen dimer.

9.5 Automating many simultaneous calculations with VASP: the Al_2O_3 system, and data analysis with MATLAB

A similar problem to what was discussed in section 9.4 which can also be solved with VASP is determining the ground state energy and equilibrium lattice parameter of a crystal. Because it is hard to predict exactly what the equilibrium lattice constant a_0 of a crystal will be, we can 'scan' a space with an arbitrary number of calculations of different atomic configurations and then use some subsequent mathematical analysis to determine the atomic configuration and/or lattice constant which minimizes the energy of the system. In this section, we will solve this type of problem for the case of an Al_2O_3 corundum crystal.

Diagrammatically, the below figure represents the general idea of what the following script attempts to demonstrate. We are taking some crystal whose unit cell is defined with lattice vectors originating at some arbitrary origin and then using BASH to automatically expand and contract that crystal by an arbitrary fraction of its equilibrium lattice parameter with an arbitrary number of calculation steps. In the figure, the black box with label a_{0i} corresponds to the equilibrium lattice parameter. The contracted and expanded lattice parameters are represented as boxes with labels a_{0c} in red and a_{0e} in blue respectively. Since we are modifying the interatomic distances in the crystal, we are therefore modifying the crystal's polarizability which has large implications for the optical properties of said crystal (a direction of my own Ph.D. research).



In this section, we discuss a script used to scan 41 unique lattice parameters in a crystal structure: 20 that represent contracted lattice parameters compared to the predicted equilibrium a_0, one for the predicted equilibrium value of a_0, and another 20 that represent expanded lattice parameters compared to the predicted equilibrium a_0. That being said however, the script is set up to calculate an arbitrary number of calculations with an arbitrary maximum and minimum identity distance for interatomic bonding. As a matter of simplicity (especially useful here in the case of the corundum structure which is actually somewhat complex), we will be modifying the bonding lengths of the crystal all at once by modifying the universal scaling factor in the the POSCAR file. In later sections (9.6, and 9.7) we will discuss iterating specific bonding lengths individually. Since there exist several comprehensive databases of crystallographic structures and information, this technique comes in very handy when you are attempting to predict new crystals, engineer atomic systems or nanocomposites, or to generate data sets to which you can fit interatomic potentials for use with molecular dynamics packages.

The following script is divided into three main sections. The first of these sections encompasses everything from the beginning of the script up to the comment "Begin VASP File Creation". As with other scripts in this text, the following line:

```
File_Name="Al2O3_automated_vasp"; mkdir ${File_Name}; cd ${File_Name}; touch ${
    File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh
    |& tee -a README.txt
```

Can be copied and pasted into the terminal and used to run the script after copying and pasting all of the script's text into the VIM editor that will be launched in the terminal. Following this, the script requests several variables to be defined like the number and type of elements, locations of executables and pseudopotentials, and parameters for running on a supercomputer cluster. In this specific instance (we will be modifying how this is handled in subsequent examples), we are handling the means of modifying individual atomic arrangements with three parameters: the initial guess lattice parameter, the number of calculation steps in the compression and expansion directions, and the maximum percent increase and decrease in the lattice parameter with respect to the initial guess.

Following the creation of these variables in section one, we use bc (see section 3.12 for more on the BASH program bc) to automatically calculate the values of three new variables: the maximum lattice parameter (maximum value of the expanded lattice constant), the minimum lattice parameter (the minimum value of the contracted lattice constant), and the step size in between each of the individual steps of the calculation. In this example, the step size is uniform, however some more sophisticated algorithms can automatically adjust the step size based on the instantaneous fineness that you are seeking in a specific region of calculation space. A small text based diagram of how this calculation is being carried out is also included after the calculations.

It is important here that bc uses a large enough number of significant figures so as to not cause strangeness with the creation of files and names later on. In this instance, I use 9 as the bc precision level. For convenience, items like the date, the number of jobs being submitted, and the value of NPAR (a VASP command for parallelization which needs to be well tuned for your system but is usually as simple as the square root of the number of computer cores that you are using) are also set as variables here. Concluding section one of this script, we create a readme file that collects most of the pertinent information from the variables that we've created for future reference.

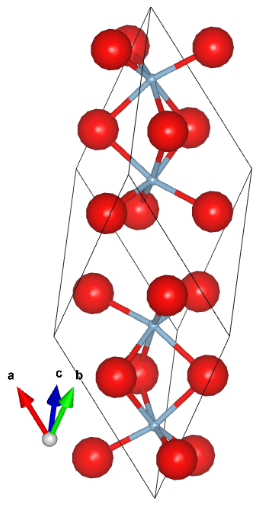
Section two of this script encompasses all code and text following the comment "Begin VASP File Creation" and up to the comment "Run VASP Calculation with SBATCH". First, we dump some lines of text to the terminal to display the status and intentions of the calculation. Next, we create a variable for the current job number, and then begin the main heavy-lifting of this script where we create a for loop to iteratively generate and submit VASP calculation scripts. The for loop we use here iterates a new variable called lat whose value at every step of the loop is determined from a sequence (which is enclosed in backticks so that it will be evaluated before anything else) starting at the minimum lattice parameter and ending at the maximum lattice parameter with a uniform step size in between all based on the previously defined variables. Starting the commands within this for loop, we create a new directory for the first step in the sequence and then navigate into that directory.

Now within this new directory, we create an SBATCH script, and VASP POSCAR, KPOINTS, INCAR, and POTCAR files using the cat command. The SBATCH script may need to be modified based on the type of cluster that you intend to use and that must be left as an exercise for the reader. Of great importance here is the POSCAR file. We are entirely handling the modification of the lattice parameter (since the corundum structure is not cubic and somewhat complicated) by setting the universal scaling factor (the second line in the POSCAR file) to be controlled by the same variable lat from the for loop. This means that at every instant (since we set the initial lattice parameter to be 1.000000 in section one of this script) the crystal's size will be modified along every basis vector by some factor between 0.98 and 1.02 with equal spacing based on the specific step in the for loop and the variables defined at the outset of this script. In later examples (see sections 9.7 and 9.6), we will individually modify atomic positions with a similar method.

Concluding this section, we create the INCAR and KPOINTS files for this calculation. These files have sensitive parameters which generally should be understood well before tinkering with. In the INCAR file, it is important to match the ISMEAR type with the type of material you are intending to calculate. Additionally, it is important to set the ENCUT value to be slightly greater than the largest ENCUT value that is given in the pseudopotentials for the individual elements that you are using. Finally in this section, we create a new POTCAR file for every step of the calculation. I find this to be the easiest way compared to more tediously statically linking every calculation to an external POTCAR file. The POTCAR file itself is created based on variables defined in section one of this script before a switch case block which tells the computer what to do in cases of 1, 2, 3, and 4 element simulations. It is straightforward to add more cases if you desire to have n elements in the simulation.

The third section of this script extends from the comment "Run VASP Calculation with SBATCH" to the end of the script. First, we print some text to the terminal that lets us know what step in the loop we are currently on and that we are about to submit a job to the computer. We then use the sbatch command to submit the job and increment the variable storing the number of the current job up by one using the expr command. Ending the for loop, we navigate up one directory and then let the loop continue until it exhausts its initial conditions upon which time the script will pass the done command and progress to the final commands. Finally, we print the status of the queue using the squeue command and set up a while loop that will run infinitely and tells the user the current status (again with the squeue command) of the calculations.

The atomic arrangement of the primitive Al_2O_3 crystal defined in the POSCAR file we create in this script can be visualized with VESTA as the following (note that this is the primitive cell and may be a slightly different view from what is normal to see for the sapphire structure, however the savings in number of atoms and therefore simplicity of calculation warrant the use of the primitive vs. the conventional unit cell in this case):



```
#!/bin/bash

# Copy and paste the below line into the cluster terminal to make and run the script
    (paste into vim and save with :wq)
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks
    to Byte Commander on Stack Exchange)

# File_Name="Al2O3_automated_vasp"; mkdir ${File_Name}; cd ${File_Name}; touch ${
   File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh
   |& tee -a README.txt

```

```
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Give the Following Variables -:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Job_Name="Al2O3_automated_vasp" # Give the name you want to apply to all files here

Description="energies of automatically expanded and contracted bond lengths in a Al2O3 crystal" # Please give a short description of the calculation for the README.txt file
Author="Steven E. Bopp, Materials Science & Engineering"

System_Name="Al2O3"      # Give a calculation title for VASP
Number_of_Elements="2"  # Number of individual elements that are in the simulation
Element_1="Al"          # Give the symbol of the element 1 in your POSCAR file
Element_2="O"           # Give the symbol of the element 2 in your POSCAR file
Element_3="n/a"         # Give the symbol of the element 3 in your POSCAR file
Element_4="n/a"         # Give the symbol of the element 4 in your POSCAR file

Precision_Level="HIGH"  # Give the level of precision that you want to use for the VASP calculation. Options are: NORMAL, MEDIUM, HIGH, LOW

Job_Time="00:10:00"     # Give the run time in hh:mm:ss
Job_Nodes="1"           # Number of nodes to use for the calculation
Job_Queue="regular"     # Calculation queue (e.g., 'debug' or 'regular')
Computer_Name="cluster" # Cluster name on which the calculation is being run

Lattice_Parameter="1.000000"              # Give the real or experimental minimum energy interatomic distance between oxygen atoms (or give 1.000000 if scaling with POSCAR scaling factor)
Calculation_Steps_in_Each_Direction="20"  # Give the total number of calculation points in each direction (e.g. 20 in the + direction, 20 in the - direction = 41 total including a0)
Lattice_Parameter_Variation="2"           # Give upper and lower bounds of % a0 change e.g., a value of 2 would mean that you want a 2% variation which is a0 +/- 0.02*a0

echo " Running VASP calculations on expanded and contracted lattice parameters of ${System_Name}"
echo " with a0=${Lattice_Parameter}A, a ${Lattice_Parameter_Variation}% maximum a0 variation, and +/- ${Calculation_Steps_in_Each_Direction} individual calculations"

Module_Name="vasp"                                                    # Give the name of the module that you want to load e.g., vasp, lammps, espresso, etc ...
Module_Location="~/uname/codes/vasp/vasp.5.3/vasp"            # Executable location
Pseudo_Location="~/uname/pseudopotentials/PBE/potpaw_PBE.54/" # Location of pseudopotential files for POTCAR, select: LDA, PBE, PW91
#Pseudo_Location="~/uname/pseudopotentials/LDA/potpaw_LDA/"   # Location of pseudopotential files for POTCAR, select: LDA, PBE, PW91

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:- BASH and bc Calculated Variables for VASP Automation :-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Lat_Param_Max=$(echo "scale=9;($Lattice_Parameter)*(     1.0 $Lattice_Parameter_Variation)" | bc)                        # Use bc to calculate a0_max
```

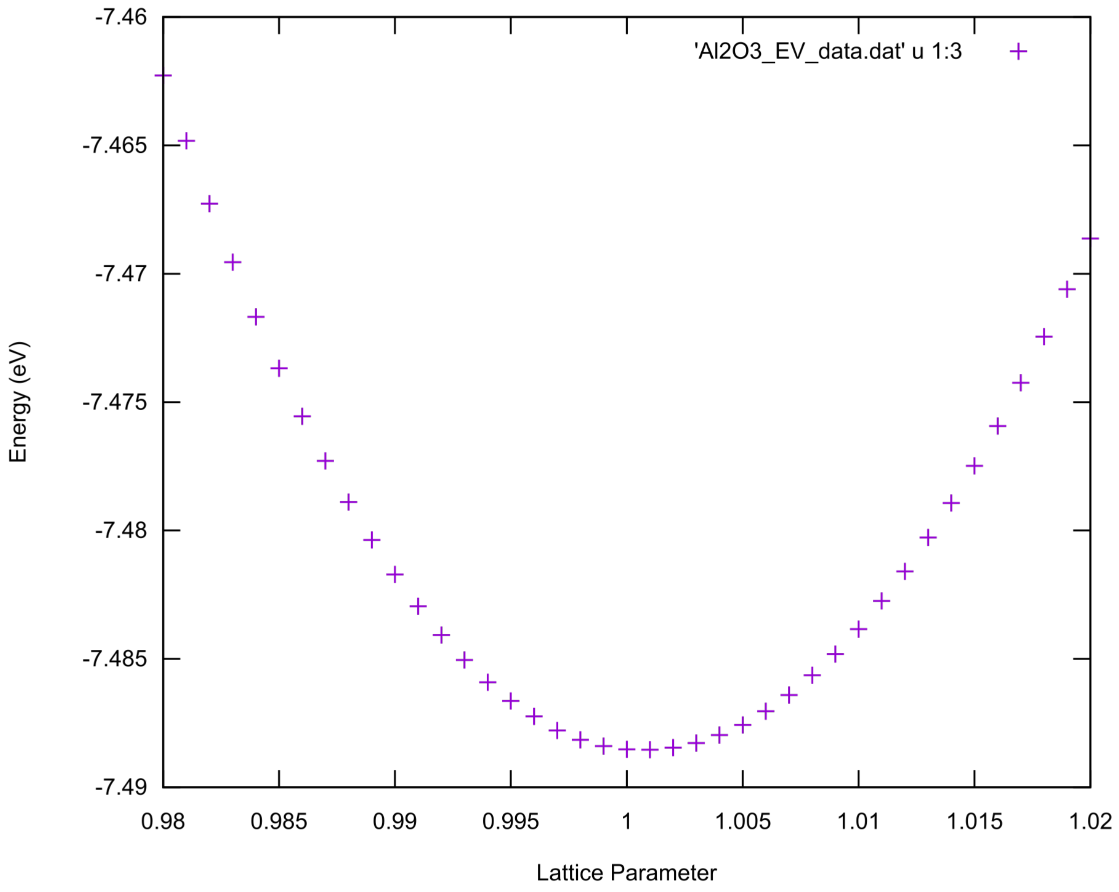
```
Lat_Param_Min=$(echo "scale=9;($Lattice_Parameter)*(1 - 0.0$Lattice_Parameter_Variation)" | bc)                       # Use bc to calculate a0_min
Step_Size=$(echo "scale=9;( (($Lat_Param_Max) - ($Lat_Param_Min))/($Calculation_Steps_in_Each_Direction*2) )" | bc) # Use bc to calculate N_steps as step size
# The above calculations determine equal step sizes between a0_min and a0_max, and are explained diagrammatically in the following:
#                  a0_min          a0           a0_max
#                    |<- N_steps ->|<- N_steps ->|
echo " Max Lattice Parameter is ${Lat_Param_Max}, Min Lattice Parameter is ${Lat_Param_Min}, Step Size is ${Step_Size}"

Number_of_Jobs=$(echo "scale=2;(($Calculation_Steps_in_Each_Direction*2)+1)" | bc)
Date=$(date '+%d/%m/%Y %H:%M:%S')            # Give date in day/month/year hr/min/sec thanks user1293137 from https://unix.stackexchange.com/
NPAR=$(echo "scale=0;sqrt($Job_Nodes)" | bc) # Calculate NPAR to be inserted into INCAR file [~sqrt(job cores)]; uses BASH program bc; scale=0 sets bc to round to nearest integer (necessary for VASP)

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF
Job Name: ${Job_Name}.sh
This is a ${Module_Name} calculation of the ${System_Name} system to calculate ${Description}.
Calculating with ${Number_of_Elements} element(s): ${Element_1}, ${Element_2}, ${Element_3}, and ${Element_4} for ${Job_Time} with ${Job_Nodes} job nodes on the ${Job_Queue} queue.
Calculated with ${Computer_Name} by ${Author} on $Date.

A transcript of the calculation as seen from the terminal follows:

EOF

echo " Writing file README.txt..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Begin VASP File Creation :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:- Automatically Generate VASP Input Files -:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:- What Files Are Generated is Controlled by the Given Lattice Vars. -:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running script ${Job_Name}.sh..."

echo " The time is currently $Date "

echo " Automatically generating input files for ${Module_Name} "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:- Begin Main for() Loop for VASP File Creation :-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
Current_Job_Number="1"

for lat in `seq -w ${Lat_Param_Min} ${Step_Size} ${Lat_Param_Max}`; do # ` is a backtick: Everything between backticks is executed by the shell before the main command, output is then used by that command

echo " The Lattice Constant Variable (lat) at this step = $lat"

mkdir ${System_Name}_a0_${lat}; cd ${System_Name}_a0_${lat} # Make and navigate into newly created directly for each iteration in the for() loop

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > ${Job_Name}_a0_${lat}.sb << EOF
#!/bin/bash
#SBATCH --job-name=${Job_Name}_a0_${lat}
#SBATCH -N ${Job_Nodes}
#SBATCH -C haswell
#SBATCH -q ${Job_Queue}
#SBATCH -t ${Job_Time}

module load ${Module_Name}
srun -n32 -c2 --cpu_bind=cores ${Module_Location}

EOF

echo " Writing input file ${Job_Name}_a0_${lat}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create POSCAR File for VASP -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > POSCAR << !
# Al2 O3 Materials Project, Primitive
${lat}
      5.17795526        0.00000000        0.00000000
      2.94847555        4.25649065        0.00000000
      2.94847555        1.54436294        3.96644119
 Al  O
     4       6
Direct
      0.14790400        0.14790400        0.14790400
      0.35209600        0.35209600        0.35209600
      0.64790400        0.64790400        0.64790400
      0.85209600        0.85209600        0.85209600
      0.94385400        0.55614600        0.25000000
      0.44385400        0.75000000        0.05614600
      0.55614600        0.25000000        0.94385400
      0.25000000        0.94385400        0.55614600
      0.75000000        0.05614600        0.44385400
      0.05614600        0.44385400        0.75000000
!

echo " Writing input file POSCAR..."
echo "  done"
```

```

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:- Create KPOINTS File for VASP :-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > KPOINTS << EOF
Automatic mesh
0              ! number of k-points = 0 -> automatic generation scheme
M
11  11  11     ! subdivisions N_1, N_2 and N_3 along recipr. lat. vectors
0  0  0        ! optional shift of the mesh (s_1, s_2, s_3)
EOF

echo " Writing input file KPOINTS..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:- Create INCAR File for VASP :-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > INCAR << EOF
# General Setup
  System = ${System_Name}      # Calculation Title
  PREC   = ${Precision_Level} # Options: Normal, Medium, High, Low

  ISMEAR = -5      # tetrahedron method with Blochl corrections
  SIGMA  = 0.04    # specifies the width of the smearing in eV
  ENCUT  = 425     # specifies the cutoff energy for the plane-wave-basis set in eV
  ALGO   = FAST    # mixture of the Davidson and RMM-DIIS algorithms

# Parallelization
  NPAR   = ${NPAR}         # approx. SQRT(number of cores)
EOF

echo " Writing input file INCAR..."
echo "  done"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file POTCAR -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# This switch/case will automatically generate a POTCAR file based on the options in
    the header of this script
# Create POTCAR with cat in the exact order that elements appear in the POSCAR file

case $Number_of_Elements in

  1) echo " One element selected for simulation: $Element_1"
     cat ${Pseudo_Location}${Element_1}/POTCAR > POTCAR ;;
  2) echo " Two elements selected for simulation: $Element_1 $Element_2"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR
     > POTCAR ;;
  3) echo " Three elements selected for simulation: $Element_1 $Element_2 $Element_3
    "
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR
     ${Pseudo_Location}${Element_3}/POTCAR > POTCAR ;;
  4) echo " Four elements selected for simulation: $Element_1 $Element_2 $Element_3
```

```
    $Element_4"
      cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR
      ${Pseudo_Location}${Element_3}/POTCAR ${Pseudo_Location}${Element_4}/POTCAR >
    POTCAR ;;

esac

echo " Writing input file POTCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Run VASP Calculation with SBATCH -:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Submitting ${Job_Name}_a0_${lat}.sb via sbatch..."

echo " Running job ${Current_Job_Number} out of ${Number_of_Jobs} job(s) on the ${
    Job_Queue} queue with ${Job_Nodes} node(s) per job for ${Job_Time} each"

sbatch ${Job_Name}_a0_${lat}.sb                              # Submit job to queue

Current_Job_Number=$(echo `expr $Current_Job_Number + 1`)  # Add 1 to the
    Current_Job_Number counter variable

echo "  Begin:"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-: End VASP File Creation -:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-: End Main for() Loop for VASP File Creation -:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cd .. # Navigate out of newly created directory

done  # End of main for() loop

#echo " The contents of this directory are now the following:" ; ls # List newly
    created directories

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:End Elapsed Time Measurement-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

squeue -u uname

echo " The time is currently $Date "

echo " Dumping terminal session into README.txt"

echo " Success, End of Script, Running squeue -u uname on a 5 Second Loop"

while [ 1 ] ; do Jobs=$(squeue -u uname | wc -l); echo "`expr ${Jobs} - 1` jobs in
    the queue"; squeue -u uname; date; sleep 5; done # continue to update the squeue
     every 5 seconds
```

Using Gnuplot (for more information, see section 3.8), we can visualize what the results of a sample calculation for this system would be as follows:



Using some mathematical analysis using MATLAB, we can also determine the absolute minimum energy value in the calculation set which corresponds to the crystal's equilibrium atomic configuration. In section 9.8 we will see how to automatically extract all of the pertinent data from these calculations, suffice it to say here however that we have this data already collected and will import it into MATLAB for analysis. Please see the following .dat file from which the above plot was created:

```
"Lattice Parameter a0" "Total Energy E0" "Energy Per Ion" "Volume Per Ion in A"
0.980000000 -.74622815E+02 -7.46228 8.23
0.981000000 -.74648247E+02 -7.46482 8.25
0.982000000 -.74672660E+02 -7.46727 8.28
0.983000000 -.74695513E+02 -7.46955 8.30
0.984000000 -.74716822E+02 -7.47168 8.33
0.985000000 -.74736787E+02 -7.47368 8.35
0.986000000 -.74755492E+02 -7.47555 8.38
0.987000000 -.74772922E+02 -7.47729 8.41
0.988000000 -.74788903E+02 -7.47889 8.43
0.989000000 -.74803654E+02 -7.48037 8.46
0.990000000 -.74817090E+02 -7.48171 8.48
0.991000000 -.74829533E+02 -7.48295 8.51
```

```
0.992000000 −.74840696E+02 −7.48407 8.53
0.993000000 −.74850368E+02 −7.48504 8.56
0.994000000 −.74859072E+02 −7.48591 8.59
0.995000000 −.74866401E+02 −7.48664 8.61
0.996000000 −.74872434E+02 −7.48724 8.64
0.997000000 −.74877908E+02 −7.48779 8.66
0.998000000 −.74881481E+02 −7.48815 8.69
0.999000000 −.74883964E+02 −7.4884 8.72
1.000000000 −.74885214E+02 −7.48852 8.74
1.001000000 −.74885396E+02 −7.48854 8.77
1.002000000 −.74884605E+02 −7.48846 8.79
1.003000000 −.74882780E+02 −7.48828 8.82
1.004000000 −.74879680E+02 −7.48797 8.85
1.005000000 −.74875675E+02 −7.48757 8.87
1.006000000 −.74870372E+02 −7.48704 8.90
1.007000000 −.74864067E+02 −7.48641 8.93
1.008000000 −.74856399E+02 −7.48564 8.95
1.009000000 −.74848102E+02 −7.48481 8.98
1.010000000 −.74838383E+02 −7.48384 9.01
1.011000000 −.74827426E+02 −7.48274 9.03
1.012000000 −.74815929E+02 −7.48159 9.06
1.013000000 −.74802749E+02 −7.48027 9.09
1.014000000 −.74789288E+02 −7.47893 9.11
1.015000000 −.74774771E+02 −7.47748 9.14
1.016000000 −.74759273E+02 −7.47593 9.17
1.017000000 −.74742507E+02 −7.47425 9.20
1.018000000 −.74724506E+02 −7.47245 9.22
1.019000000 −.74705980E+02 −7.4706 9.25
1.020000000 −.74686281E+02 −7.46863 9.28
```

Importing this data (the first and third columns) into MATLAB, we can use splines and the 'find min' functionality to find the minimum energy configuration based on interpolation:

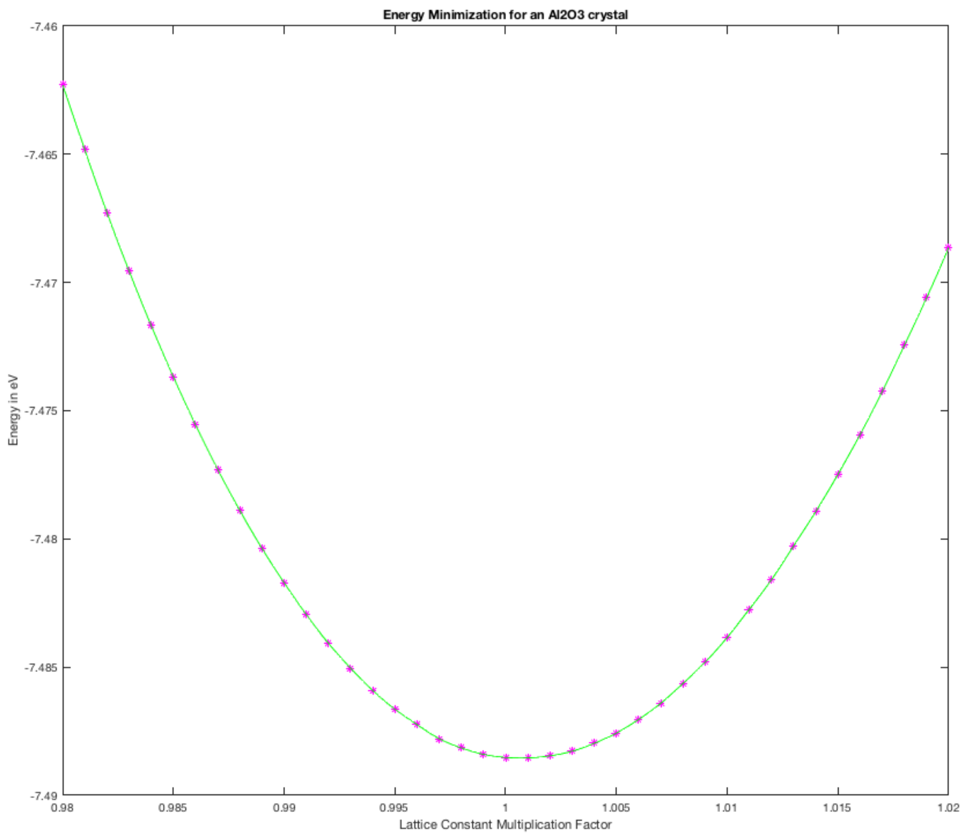
```
%% Al2O3_Lattice_Constant.m
%  Written by Steven E. Bopp

%%
Lattice_Parameter=[0.980000000
0.981000000
0.982000000
0.983000000
0.984000000
0.985000000
0.986000000
0.987000000
0.988000000
0.989000000
0.990000000
0.991000000
0.992000000
0.993000000
0.994000000
0.995000000
0.996000000
0.997000000
0.998000000
0.999000000
1.000000000
1.001000000
```

```
1.002000000
1.003000000
1.004000000
1.005000000
1.006000000
1.007000000
1.008000000
1.009000000
1.010000000
1.011000000
1.012000000
1.013000000
1.014000000
1.015000000
1.016000000
1.017000000
1.018000000
1.019000000
1.020000000];

Energy_Per_Ion=[ -7.46228
 -7.46482
 -7.46727
 -7.46955
 -7.47168
 -7.47368
 -7.47555
 -7.47729
 -7.47889
 -7.48037
 -7.48171
 -7.48295
 -7.48407
 -7.48504
 -7.48591
 -7.48664
 -7.48724
 -7.48779
 -7.48815
 -7.4884
 -7.48852
 -7.48854
 -7.48846
 -7.48828
 -7.48797
 -7.48757
 -7.48704
 -7.48641
 -7.48564
 -7.48481
 -7.48384
 -7.48274
 -7.48159
 -7.48027
 -7.47893
 -7.47748
 -7.47593
 -7.47425
 −7.47245
 −7.4706
 −7.46863];

%%

E0=Energy_Per_Ion';
x=Lattice_Parameter'; xx=[0.980000000:.0001:1.020000000];
yy=spline(x,E0,xx);         % Create spline interpolate for the lattice constant

figure(1);
plot(x,E0,'m*',xx,yy,'g'); % Plot spline interpolate and Energy with Lattice
    Parameter
xlabel('Lattice Constant Multiplication Factor'); ylabel('Energy in eV');
title('Energy Minimization for an Al2O3 crystal');
xlim([0.980000000 1.020000000])
% Pentagram p; hexagram h; diamond d; square s;

indexmin=find(min(yy) == yy);                 % Define indexmin
xmin = xx(indexmin); ymin = yy(indexmin);    % Calculate minimum values
A0 = xmin;
fprintf('Lattice Constant Multiplication Factor for Al2O3:%g \n',A0) % Display
    lattice constant and error from expected value

```

MATLAB should report the following:

```
Lattice Constant Multiplication Factor for Al2O3:1.0007
```

MATLAB should also generate the following plot of what it has fit with the spline interpolation:



What this specifically means is that, from the spline interpolation, MATLAB has calculated that 1.0007 times the size (in all basis vectors) of the crystal given in the script's POSCAR file should be the equilibrium atomic configuration.

9.6 Adsorption of an AlO dimer on a c-axis oriented Al_2O_3 surface, and data analysis with MATLAB

In a previous section (9.4), we discussed using the shell to automatically create an arbitrary number of calculation scripts for the ground state energy of an O_2 dimer with varying bond length. In the present section of this text, we will extend this concept to a crystal of Al_2O_3 with the corundum structure (which, recall, is trigonal and not hexagonal: a common misconception) where there is an incoming AlO dimer which is being adsorbed onto the surface of the crystal.

At this point in the text, many of these scripts may be becoming somewhat familiarly constructed. This is true and a direct effect of my own efforts in creating scripts which are as quick to modify, highly modular, and feature-packed as reasonable for a given application. I find that this method of doing things assists greatly in dynamic and flexible multi-use tools for your own research which can be quickly modified and then rapidly deployed to represent more or less any atomic arrangement that you can reasonably imagine. In the following paragraphs, I will describe the main parts of the included script whose intention is to calculate the ground state energy of an arbitrary number of atomic configurations in the corundum Al_2O_3 system with an AlO dimer approaching its surface.

In the beginning of the script, I include the small functionality where I copy and paste the following line into the terminal:

```
File_Name="Slab_Al2O3_AlO_Dimer_vertical_Al_up_on_O_automated_vasp_11x11x1"; mkdir $
    {File_Name}; cd ${File_Name}; touch ${File_Name}.sh; chmod +x ${File_Name}.sh;
    vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt
```

Just like in section 7.3, as well as many others, I built in this functionality in order to automate the creation of a shell script file, making it executable, then launching vim (see section 3.7 for more on vim) so that you can copy and paste your script into the newly creates shell script file, execute the script, and then use tee to copy everything printed to the terminal into a README file. Additionally, since I take large advantage of the flexibility of this general format of script, modifying it hundreds of times with small tweaks, I also include a small graphic in ASCII text (see more on these graphics in section 12.2) to remind myself at a glance exactly what the calculation is attempting to do.

The first main section of the script encompasses everything following the comment "Give the Following Variables" and up to the comment "Begin VASP File Creation". In this section, we begin with defining many optional variables for things like VASP options (e.g., the precision level, the type of pseudopotential I want to use, and the k-grid size), the cluster queue and processor architecture that I select for my calculation, and a few other parameters like whether I want to submit the calculations in chunks rather than all at once, how long I want to wait between requesting an update on the calculation status on the cluster, and whether or not the calculations will actually be submitted using squeue or if that step will be skipped (this is handy if you are just testing to make sure that you are generating the proper number of files and that they are all placed where you are expecting).

Continuing in the first section, we next use these previously defined variables as well as a few switch/case statements and the bc program (for more on bc, see 3.12) to define the locations of our executables, choose pertinent numbers of cores and threads that we want to use for our calculation, defining what the script will do in the event of a test option vs. a full calculation (here I also define sequences for the numbers of calculations that will be done, e.g., for the 'full' calculation, we will be running new calculations every 0.05 angstroms for diatomic molecule and crystal surface separations between 0.05 and 3 angstroms), defining where the pseudopotential files are located, defining what to do for the different queue options which can be given by the script, and defining the date as well as the VASP INCAR option NPAR. NPAR is a parallelization parameter given to VASP in the INCAR file which is usually given as the square root of the number of computer cores you're using and needs to be carefully defined and tested. In my own tests, this works best for numbers of processors that are perfect squares. A properly defined NPAR will substantially increase the speed of convergence for a calculation in most cases. Finally in this first section, we create a readme file with much of the pertinent information given in the present section reiterated for the sake of future reference.

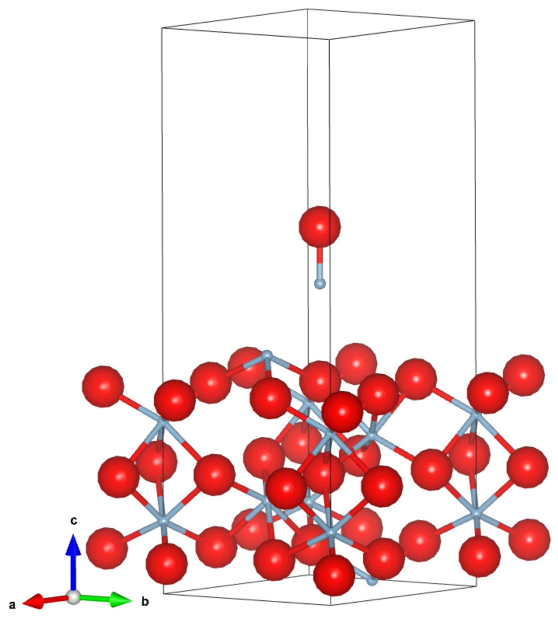
Section two is everyting after the comment "Begin VASP File Creation" and before the comment "Create SBATCH Script". Here, initially, we print lots of pertinent information to the terminal based on what we are giving the program to calculate. We initially define a variable for the current job number which will be useful later on in the calculations. After that, we begin the main, heavy-lifting part of the script which is a for loop. This for loop creates an iterated variable called lat which is based on the sequence given in the Calculation_Type variable. Everything after this for loop and up to the "done" command is given later on will be iterated an arbitrary number of times based on the sequences given in the Calculation_Type switch/case statement above. This loop is kicked off by the creation of a new directory named with rules based on the current lattice parameter in the sequence, and then the script enters that directory.

Directly after this, we begin section three of the script where we create create an SBATCH file, as well as POSCAR, KPOINTS, INCAR, and POTCAR files for every point in the calculation. Within the SBATCH script, I automate the placement of things like executable location(s), and what queue, architecture, and job times to use based on the variables given in section one. In the POSCAR file, I have used VESTA and Atomsk (for more on VESTA Atomsk see sections 8.3, and 8.5) to create a crystal where there are atoms located at distances a and b (defined by variables lat_a, and lat_b above the surface. Here, lat_a, and lat_b are calculated automatically based on the variable Aluminum_Oxygen_Dimer_Bond_Length which can be found in literature or calculated for one's self in a similar way to what is defined in section 9.4 where we calculated the equilibrium bonding distance of an oxygen dimer. This arrangement of atoms is shown within the ASCII text diagram following the file creation (again, I find these little diagrams extremely helpful for reminding myself what is happening at different steps in a calculation).

Rounding out section three of the script, we create three more files. The first of these is the KPOINTS VASP file which we make based on recommendations from the VASP documentation and examples. Next, we create an INCAR file (note how we do not have the ISPIN defined since we do not care about spin polarization in this case). Finally, completing section three here, we create a POTCAR file. The POTCAR file is created based on the number of separate atomic species that were defined previously and a switch/case statement governs what number of pseudopotentials will be used in the POTCAR file. I find this to be substantially easier than trying to link to a static POTCAR file or make the file by hand each time because I frequently change the ordering of atoms when I tweak POSCAR files for slightly different systems, and reducing the number of things that I actually need to modify for each calculation reduces the number of human-introduced errors (since I'm not perfect and frequently make mistakes when it's a question of tedious repetition or slight tweaks on very large numbers of files).

Finally, section five encompasses everything following the comment "Run VASP Calculation with SBATCH" through to the end of the script. Initially, we print some text to the terminal to know that we have reached this section in the script without error. Next, we define what the script should do in response to various user-defined variables given in section one of the script such as whether the job will be submitted as a full calculation or a test, and whether the calculations will be submitted all at once (you may use this if you are submitting to a normal calculation queue) or submitted in chunks (useful if you are submitting to a queue which limits the number of total concurrent jobs which you can run at a single time). After that, since we have only completed one single step of the for loop up to this point, we navigate up one directory and then the for loop begins again with the next element in the sequence. The for loop continues until its conditions have been exhausted. Upon exhausting those conditions, the queue is printed to the terminal as well as some text about the status of the calculation, and a while loop begins printing the status of the calculations at a given time based on a variable defined in section one above.

The atomic arrangement defined in the POSCAR file we create in this script can be visualized with VESTA as the following:



Please see the following script:

```
#!/bin/bash

# Copy and paste the below line into the cluster terminal to make and run the script
    (paste into vim and save with :wq)
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks
    to Byte Commander on Stack Exchange)

# File_Name="Slab_Al2O3_AlO_Dimer_vertical_Al_up_on_O_automated_vasp_11x11x1"; mkdir
    ${File_Name}; cd ${File_Name}; touch ${File_Name}.sh; chmod +x ${File_Name}.sh;
    vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt

#                             +--------+
#                             |   Al   | ^   Dimer Atom 1
#                             +--------+ |
#                                        |   (1.6337A)
#                             +--------+ |
#                             |   O    | <   Dimer Atom 2
#                             +--------+ |
#                                        |   Variable z-coord.
#                             +--------+ |
#                             |   O    | v   C- Sapphire Slab Surface
#                             +--------+

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Give the Following Variables -:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
Job_Name="Slab_Al2O3_AlO_Dimer_Ti_up_on_O_automated_vasp_11x11x1" # Give the name
    you want to apply to all files here

Description="energies of automatically ranged and vertically arranged AlO_Dimer with
     the Ti atom facing up approaching an O atom (away from the surface) bond
    lengths on a Al2O3 c-axis surface" # Please give a short description of the
    calculation for the README.txt file
Author="Steven E. Bopp, Materials Science & Engineering"

System_Name="Slab_Al2O3_AlO_Dimer_Ti_up_on_O_11x11x1"      # Give a calculation title
     for VASP

Number_of_Elements="2"  # Give the number of individual elements that are in the
    simulation
Element_1="Al"          # Give the symbol of the element 1 in your POSCAR file
Element_2="O"           # Give the symbol of the element 2 in your POSCAR file
Element_3="n/a"         # Give the symbol of the element 3 in your POSCAR file
Element_4="n'a"         # Give the symbol of the element 4 in your POSCAR file

Precision_Level="HIGH"   # VASP level of precision: NORMAL, MEDIUM, HIGH, LOW
KPOINTS_Grid="11 11 1"   # KGrid (give as wanted in the KPOINTS file e.g., '9 9 9')
Job_Time="02:01:00"      #Run time in hh:mm:ss
Job_Time_Min="00:39:59"  # Minimum run time of the script
Job_Nodes="1"            # Nodes that you want to use for the calculation
Job_Queue="flex"         # Calculation queue (e.g., 'debug', 'regular', 'flex', or '
    low') low and flex queues have 50 and 75% discounts respectively for KNL
Architecture="knl"       # 'haswell' for Haswell chips or 'knl' for Knight's Landing
    )
Computer_Name="cluster"  # Give the name of the cluster
Module_Name="vasp"       # Module to load e.g., vasp, lammps, espresso, etc...
PP_Type="PBE"            # Pseudopotential type to load  e.g., PBE or LDA
Calculation_Type="full"  # Calculation type (e.g., 'test' will run 1 point and 'full
    ' will run all of the points)
Submission_Type="1"      # 0 will NOT submit via sbatch and 1 WILL submit via sbatch
Piecewise_Submission="0" # 0 will NOT submit piecewise and 1 will submit piecewise,
    sleeping for 15 seconds between submissions)
Piecewise_Sleep="15"     # Give the number of seconds between job submission(s) for
    the Piecewise_Submission case (recommended to be ~30 to 50% of the run-time of a
     VASP calculation)
SQS_Update_Time="30"     # Give the number of seconds that you want the final squeue
     update cycle to wait between updates (recommended value: '30')

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:- BASH and bc Calculated Variables for VASP Automation :-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

case $Architecture in             # This switch/case will automatically switch the
    environments and executable locations for various architectures like haswell and
     knl
  haswell) echo " Haswell architecture selected: requires Intel PrgEnv"
           Processes_Per_Node="32" ;Threads_Per_Process="2"
      # Recommended Haswell settings for 1 node and 32 MPI processes per node with
    2 threads each
           Module_Location="~/uname/codes/vasp/vasp.5.3/vasp"         ;; #
    Executable location for HSW
  knl)     echo " Knight's Landing architecture selected
           Processes_Per_Node="64" ;Threads_Per_Process="4"
      # Recommended MPI setting for 1 node. NPAR should be sqrt(Processes_Per_Node)
```

```
!! NPAR = 8 gives substantial savings of ~67% compared to NPAR = 1
          Module_Location="~/uname/vasp_cray_compiled/vasp.5.3/vasp" ;; # Executable location for KNL
esac

case $Calculation_Type in          # This switch/case will automatically switch the calculation sequence based on the $Calculation_Type variable
  test)    echo " Running a test calculation"
           Sequence=$(seq -w 1 1 1); Number_of_Jobs=`echo $Sequence | wc -w` ;; # Single point calculation sequence
  full)    echo " Running a full calculation"
           Sequence=$(seq -w 0.05 0.05 3.0); Number_of_Jobs=`echo $Sequence | wc -w` ;; # Multiple point calculation sequence
esac

case $PP_Type in                   # This switch/case will automatically switch the pseudopotential type based on the $PP_Type variable
  PBE)    echo " Using the VASP PBE.54 pseudopotentials"
          Pseudo_Location="~/uname/pseudopotentials/PBE/potpaw_PBE.54/" ;; # Location of pseudopotential files for POTCAR, select: LDA, PBE, PW91
  LDA)    echo " Using the VASP LDA pseudopotentials"
          Pseudo_Location="~/uname/pseudopotentials/LDA/potpaw_LDA/"    ;; # Location of pseudopotential files for POTCAR, select: LDA, PBE, PW91
esac

case $Job_Queue in                 # This switch/case will automatically insert options into the sbatch file
  flex)    Time_Min="#SBATCH --time-min=${Job_Time_Min}" ;; # Inserts #SBATCH --time-min=0:30:00 into sbatch
  shared)  Shared="#SBATCH --shared"                     ;; # Inserts #SBATCH --shared into sbatch
esac

Date=$(date '+%d/%m/%Y %H:%M:%S') # Give date in day/month/year hr/min/sec thanks user1293137 from https://unix.stackexchange.com/
NPAR=$(echo "scale=0;sqrt($Processes_Per_Node)" | bc) # Calculate NPAR to be inserted into INCAR file [~sqrt(job cores)]; uses BASH program bc; scale=0 sets bc to round to nearest integer (necessary for VASP)

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF
Job Name: ${Job_Name}.sh
This is a ${Module_Name} calculation of the ${System_Name} system to calculate ${Description} with ${PP_Type} pseudopotentials on a ${KPOINTS_Grid} grid..
Calculating with ${Number_of_Elements} element(s): ${Element_1}, ${Element_2}, ${Element_3}, and ${Element_4} for ${Job_Time} with ${Job_Nodes} job nodes on the ${Job_Queue} queue.
Using ${Architecture} node(s) with ${Processes_Per_Node} processes per node and ${Threads_Per_Process} threads per node for a maximum of ${Job_Time} hh:mm:ss.
Calculated with ${Computer_Name} by ${Author} on $Date.

A transcript of the calculation as seen from the terminal follows:

EOF

```

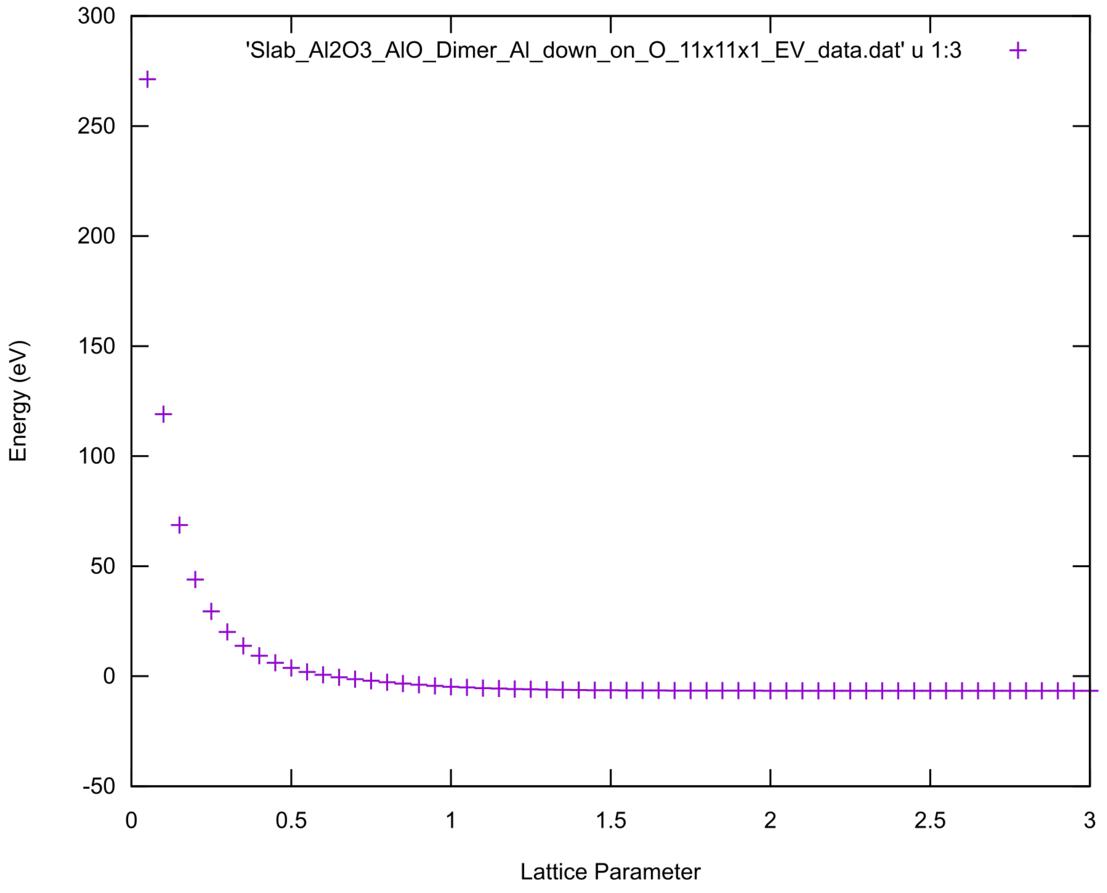
```
echo " Writing file README.txt..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Begin VASP File Creation :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:- Automatically Generate VASP Input Files -:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:- What Files Are Generated is Controlled by the Given Lattice Vars. -:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running a ${Module_Name} calculation of the ${System_Name} system to
    calculate ${Description} with ${PP_Type} pseudopotentials on a ${KPOINTS_Grid}
    grid."

echo " Calculating with ${Number_of_Elements} element(s): ${Element_1}, ${Element_2
    }, ${Element_3}, and ${Element_4} for ${Job_Time} with ${Job_Nodes} job nodes on
     the ${Job_Queue} queue."

echo " Using ${Architecture} node(s) with ${Processes_Per_Node} processes per node
    and ${Threads_Per_Process} threads per node for a maximum of ${Job_Time} hh:mm:
    ss"

echo " Calculated with ${Computer_Name} by ${Author} on $Date."

echo " Running script ${Job_Name}.sh..."

echo " The time is currently $Date "

echo " Automatically generating input files for ${Module_Name} "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:- Begin Main for() Loop for VASP File Creation :-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Current_Job_Number="1" # Begin a variable to use in the for() loop to count the
    current job up to the final job

for lat in ${Sequence}; do # Single calculation

echo " The Lattice Constant Variable (lat) at this step = $lat"

mkdir ${System_Name}_a0_${lat}; cd ${System_Name}_a0_${lat} # Make and navigate into
     newly created directly for each iteration in the for() loop

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > ${Job_Name}_a0_${lat}.sb << EOF
#!/bin/bash
#SBATCH --job-name=${Job_Name}_a0_${lat}
#SBATCH -N ${Job_Nodes}
#SBATCH -C ${Architecture}
#SBATCH -q ${Job_Queue}
#SBATCH -t ${Job_Time}
${Time_Min}
${Shared}
```

```
#OpenMP settings:
export OMP_NUM_THREADS=1
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

module load ${Module_Name}
srun -n ${Processes_Per_Node} -c ${Threads_Per_Process} --cpu_bind=cores ${Module_Location}

EOF

echo " Writing input file ${Job_Name}_a0_${lat}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create POSCAR File for VASP -:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Top_Atom="5.40425600"
Aluminum_Oxygen_Dimer_Bond_Length="1.6337"

lat_a=$(echo "scale=9; $Top_Atom + $lat" | bc)
lat_b=$(echo "scale=9; $lat_a + $Aluminum_Oxygen_Dimer_Bond_Length" | bc)

cat > POSCAR << !
# Al2O3 c-axis slab with vertically-oriented O2 dimer approaching File generated with Atomsk by ubuntu-budgie on 2021-09-23 10:57:51
1.000000
      4.75800000         0.00000000         0.00000000
     -2.37900000         4.12054887         0.00000000
      0.00000000         0.00000000        15.00000000
 Al   O
 7    10
Cartesian
 0.00000000         0.00000000         4.57283200
 0.00000000         0.00000000         1.92266800
 2.38137900         1.37214277         6.24867100
 2.38137900         1.37214277         2.40333500
 0.00237900         2.74840610         0.23383800
 0.00237900         2.74840610         4.07917400
 0.73035300         1.48751814         ${lat_b}
 0.73035300         1.48751814         ${lat_a}
 1.45594800         0.00000000         3.24775000
-0.72797400         1.26088795         3.24775000
 1.65102600         2.85966092         3.24775000
 0.92543100         1.37214277         1.07825300
 3.10935300         0.11125482         1.07825300
 3.10935300         2.63303073         1.07825300
 0.73035300         1.48751814         5.40425600
 0.73035300         4.00929405         5.40425600
-1.45356900         2.74840610         5.40425600
!

echo " Writing input file POSCAR..."
echo "  done"

#     +-------+
#    |Atom_1| ^ Dimer Atom 1 <—— Ti in this case is pointing up
#    +——————+ |
#             |lat_b=lat_a + dimer equilibrium bond length
#    +——————+ |
#    |Atom_2| v Dimer Atom 2 <—— N in this case is pointing down
#    +——————+ |
#             |lat_a=Surface z—coord. + variable z—coord.
#    +——————+ |
#    |Atom_3| v Slab Surface
#    +——————+

#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:— Create KPOINTS File for VASP :—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—

# ~ 1000 k—points per atom for metals will reduce error to approximately 10 meV; ~
   100 k—points per atom for insulators

cat > KPOINTS << EOF
Automatic mesh
0                    ! number of k—points = 0 —> automatic generation scheme
M
${KPOINTS_Grid}      ! subdivisions N_1, N_2 and N_3 along recipr. latt. vectors
0  0  0              ! optional shift of the mesh (s_1, s_2, s_3)
EOF

echo " Writing input file KPOINTS..."
echo "  done"

#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:— Create INCAR File for VASP :—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—

# This INCAR is specific for semiconductors and insulators!

cat > INCAR << EOF
# General Setup
  System = ${System_Name}      # Calculation Title
  PREC   = ${Precision_Level} # Options: Normal, Medium, High, Low

  ISMEAR = —5       # tetrahedron method with Blochl corrections
  SIGMA  = 0.04     # specifies the width of the smearing in eV
  ENCUT  = 425      # specifies the cutoff energy for the plane—wave—basis set in eV
  ALGO   = FAST     # mixture of the Davidson and RMM—DIIS algorithms

# Parallelization
  NPAR   = ${NPAR}         # approx. SQRT(number of cores)
EOF

echo " Writing input file INCAR..."
echo "  done"


#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—: Create file POTCAR —:—:—:—:—:—:—:—:—:—:—:—:—
#—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—:—

# This switch/case will automatically generate a POTCAR file based on the options in
```

```
    the header of this script
# Create POTCAR with cat in the exact order that elements appear in the POSCAR file

case $Number_of_Elements in

  1) echo " One element selected for simulation: $Element_1"
     cat ${Pseudo_Location}${Element_1}/POTCAR > POTCAR ;;
  2) echo " Two elements selected for simulation: $Element_1 $Element_2"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR
     > POTCAR ;;
  3) echo " Three elements selected for simulation: $Element_1 $Element_2 $Element_3
    "
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR
     ${Pseudo_Location}${Element_3}/POTCAR > POTCAR ;;
  4) echo " Four elements selected for simulation: $Element_1 $Element_2 $Element_3
    $Element_4"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR
     ${Pseudo_Location}${Element_3}/POTCAR ${Pseudo_Location}${Element_4}/POTCAR >
    POTCAR ;;

esac

echo " Writing input file POTCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Run VASP Calculation with SBATCH -:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Submitting ${Job_Name}_a0_${lat}.sb via sbatch..."

echo " Running job ${Current_Job_Number} out of ${Number_of_Jobs} job(s) on the ${
    Job_Queue} queue with ${Job_Nodes} node(s) per job for ${Job_Time} each"

case $Submission_Type in # This switch/case will automatically switch whether to or
    not to submit job(s) to the queue
  0)    echo " Not submitting job(s)"   ;; # Not submitting jobs to queue
  1)    sbatch ${Job_Name}_a0_${lat}.sb ;; # Submit job to queue
esac

Current_Job_Number=$(echo `expr $Current_Job_Number + 1`)  # Add 1 to the
    Current_Job_Number counter variable

case $Piecewise_Submission in                     # This switch/case will
    automatically switch whether to or not to submit job(s) to the queue
  0)                                           ;; # Not submitting jobs to queue
  1)    squeue -u uname | head; sleep ${Piecewise_Sleep} ;; # Submit job to queue if
     trying to run batches for debugging on debug queue
esac

echo "  Begin:"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: End VASP File Creation -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-: End Main for() Loop for VASP File Creation -:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

```

```
cd .. # Navigate out of newly created directory

done  # End of main for() loop

#echo " The contents of this directory are now the following:" ; ls # List newly
    created directories

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:End Elapsed Time Measurement-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

squeue -u uname

echo " The time is currently $Date "

echo " Dumping terminal session into README.txt"

echo " Success, End of Script, Running squeue -u uname on a ${SQS_Update_Time}
    Second Loop"

while [ 1 ] ; do Jobs=$(squeue -u uname | wc -l); echo "`expr ${Jobs} - 1` jobs in
    the queue"; squeue -u uname; date; sleep ${SQS_Update_Time}; done # continue to
    update the squeue every 5 seconds
```

Using Gnuplot (for more information, see section 3.8), we can visualize what the results of a sample calculation for this system would be as follows:



Using some mathematical analysis with MATLAB, we can also determine what the absolute minimum energy value in the calculation set which corresponds to the equilibrium configuration. In section 9.8 we will see how to automatically extract all of the pertinent data from these calculations, suffice it to say here however that we have this data already collected and we will import it into MATLAB for analysis. Please see the following .dat file from which the above plot was created:

```
"Lattice Parameter a0" "Total Energy E0" "Energy Per Ion" "Volume Per Ion in A"
0.05 0.46116276E+04 271.272 17.30
0.10 0.20243127E+04 119.077 17.30
0.15 0.11681642E+04 68.7155 17.30
0.20 0.74677299E+03 43.9278 17.30
0.25 0.50043335E+03 29.4373 17.30
0.30 0.34221275E+03 20.1302 17.30
0.35 0.23460743E+03 13.8004 17.30
0.40 0.15871279E+03 9.33605 17.30
0.45 0.10387018E+03 6.11001 17.30
0.50 0.63538060E+02 3.73753 17.30
0.55 0.33390952E+02 1.96417 17.30
0.60 0.10390932E+02 0.611231 17.30
0.65 -.76796481E+01 -0.451744 17.30
```

```
0.70 −.22501286E+02 −1.32361 17.30
0.75 −.35289600E+02 −2.07586 17.30
0.80 −.46756522E+02 −2.75038 17.30
0.85 −.57108427E+02 −3.35932 17.30
0.90 −.66313806E+02 −3.90081 17.30
0.95 −.74321297E+02 −4.37184 17.30
1.00 −.81151502E+02 −4.77362 17.30
1.05 −.86891251E+02 −5.11125 17.30
1.10 −.91662102E+02 −5.39189 17.30
1.15 −.95594064E+02 −5.62318 17.30
1.20 −.98811401E+02 −5.81244 17.30
1.25 −.10142729E+03 −5.96631 17.30
1.30 −.10354201E+03 −6.09071 17.30
1.35 −.10524328E+03 −6.19078 17.30
1.40 −.10660731E+03 −6.27102 17.30
1.45 −.10769932E+03 −6.33525 17.30
1.50 −.10857432E+03 −6.38672 17.30
1.55 −.10927762E+03 −6.4281 17.30
1.60 −.10984591E+03 −6.46152 17.30
1.65 −.11030856E+03 −6.48874 17.30
1.70 −.11068832E+03 −6.51108 17.30
1.75 −.11100348E+03 −6.52962 17.30
1.80 −.11126826E+03 −6.54519 17.30
1.85 −.11149297E+03 −6.55841 17.30
1.90 −.11168558E+03 −6.56974 17.30
1.95 −.11185189E+03 −6.57952 17.30
2.00 −.11199665E+03 −6.58804 17.30
2.05 −.11212320E+03 −6.59548 17.30
2.10 −.11223413E+03 −6.60201 17.30
2.15 −.11233160E+03 −6.60774 17.30
2.20 −.11241719E+03 −6.61278 17.30
2.25 −.11249227E+03 −6.61719 17.30
2.30 −.11255797E+03 −6.62106 17.30
2.35 −.11261511E+03 −6.62442 17.30
2.40 −.11266453E+03 −6.62733 17.30
2.45 −.11270721E+03 −6.62984 17.30
2.50 −.11274344E+03 −6.63197 17.30
2.55 −.11277475E+03 −6.63381 17.30
2.60 −.11280069E+03 −6.63533 17.30
2.65 −.11282122E+03 −6.63654 17.30
2.70 −.11283797E+03 −6.63753 17.30
2.75 −.11284984E+03 −6.63823 17.30
2.80 −.11285782E+03 −6.6387 17.30
2.85 −.11286205E+03 −6.63894 17.30
2.90 −.11286265E+03 −6.63898 17.30
2.95 −.11286000E+03 −6.63882 17.30
3.00 −.11285430E+03 −6.63849 17.30
```

Importing this data (the first and third columns) into MATLAB, we can use splines and the 'find min' functionality to find the minimum energy configuration based on interpolation:

```
%% AlO_Dimer_Approaching_Al2O3.m
%  Written by Steven E. Bopp

%%
Lattice_Parameter=[0.05
0.10
0.15
0.20
```

```
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
1.05
1.10
1.15
1.20
1.25
1.30
1.35
1.40
1.45
1.50
1.55
1.60
1.65
1.70
1.75
1.80
1.85
1.90
1.95
2.00
2.05
2.10
2.15
2.20
2.25
2.30
2.35
2.40
2.45
2.50
2.55
2.60
2.65
2.70
2.75
2.80
2.85
2.90
2.95
3.00];

Energy_Per_Ion=[ 271.272
```

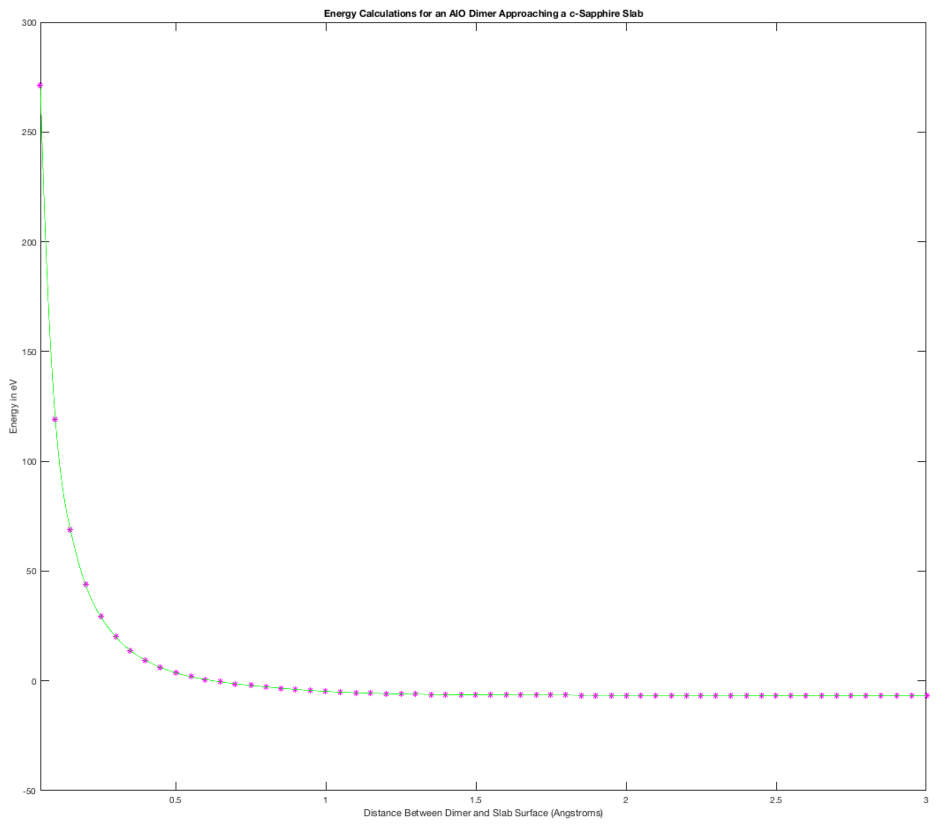
```
119.077
68.7155
43.9278
29.4373
20.1302
13.8004
9.33605
6.11001
3.73753
1.96417
0.611231
−0.45174
−1.32361
−2.07586
−2.75038
−3.35932
−3.90081
−4.37184
−4.77362
−5.11125
−5.39189
−5.62318
−5.81244
−5.96631
−6.09071
−6.19078
−6.27102
−6.33525
−6.38672
−6.4281
−6.46152
−6.48874
−6.51108
−6.52962
−6.54519
−6.55841
−6.56974
−6.57952
−6.58804
−6.59548
−6.60201
−6.60774
−6.61278
−6.61719
−6.62106
−6.62442
−6.62733
−6.62984
−6.63197
−6.63381
−6.63533
−6.63654
−6.63753
−6.63823
−6.6387
−6.63894
−6.63898
−6.63882
```

```
 -6.63849];

%%

E0=Energy_Per_Ion';
x=Lattice_Parameter'; xx=[0.05:.0001:3.00];
yy=spline(x,E0,xx);        % Create spline interpolate for the lattice constant

figure(1);
plot(x,E0,'m*',xx,yy,'g'); % Plot spline interpolate and Energy with Lattice
    Parameter
xlabel('Distance Between Dimer and Slab Surface (Angstroms)'); ylabel('Energy in eV'
    );
title('Energy Calculations for an AlO Dimer Approaching a c-Sapphire Slab');
xlim([0.05 3.00])
% Pentagram p; hexagram h; diamond d; square s;

indexmin=find(min(yy) == yy);                 % Define indexmin
xmin = xx(indexmin); ymin = yy(indexmin);    % Calculate minimum values
A0 = xmin;
fprintf('Minimum Energy Configuration Length:%g \n',A0) % Display lattice constant
    and error from expected value
```

MATLAB should report the following:

```
Minimum Energy Configuration Length:2.8852
```

MATLAB should also generate the following plot of what it has fit with the spline interpolation:



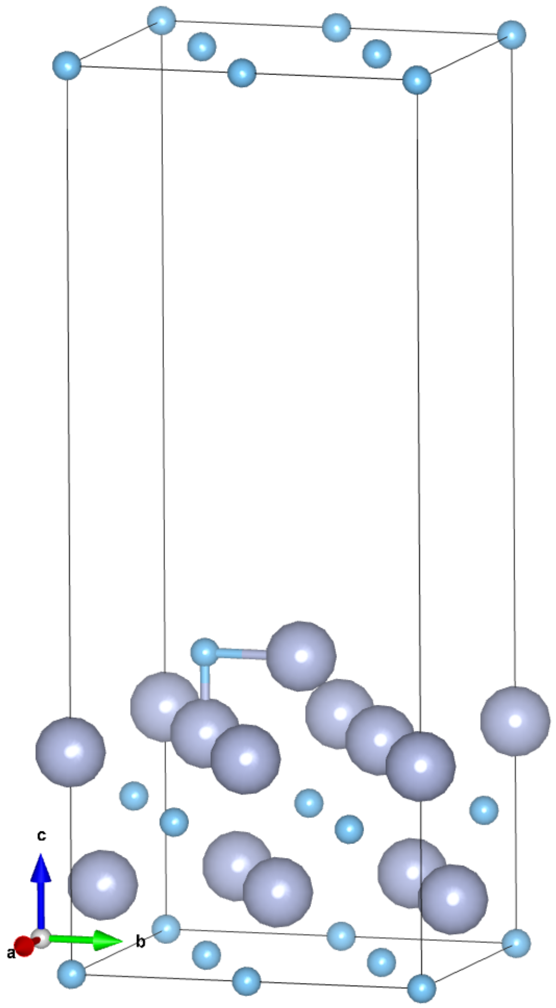
What this specifically means is that, from the spline interpolation, MATLAB has calculated that a distance of 2.8852 angstroms for this atomic configuration of the Al-side of an Al-O dimer approaching an O atom on a c-Sapphire surface is the minimum energy configuration and therefore corresponds to where a bond will form.

9.7 Adsorption of a TiN dimer on a (111)-oriented TiN Surface, and data analysis with MATLAB

Similar to section 9.6, here we will briefly discuss another orientation of dimer adsorption onto a crystal surface, e.g., where a dimer is oriented parallel (instead of orthogonal like in section 9.6) to the crystal's surface. In this case, we will be considering a nitrogen terminated slab of 111-oriented TiN with a horizontally-oriented TiN dimer approaching the slab. This calculation script is very similar to that given in section 9.6 and I will not cover it in as much detail. Refer to section 9.6 for more detail on the specific sections of this script if that is what you desire.

I will however cover the main differences in the POSCAR file since therein is the main difference between this script and the one from section 9.6. This time we create a single variable called lat_a which defines the distance that both atoms in the dimer have away from the crystal slab's surface. The distance between the two atoms is defined explicitly in the atomic positions (x, and y in the POSCAR file) with knowledge of the dimer's equilibrium bonding length from other VASP calculations similar to those explained in section 9.4. As with before, small ASCII diagrams at the beginning of the script and within the section responsible for writing the POSCAR file aid in quick recognition of exactly what is being calculated at a glance.

The atomic arrangement defined in the POSCAR file we create in this script can be visualized with VESTA as the following:



Please see the following script:

```
#!/bin/bash

# Copy and paste the below line into the cluster terminal to make and run the script
    (paste into vim and save with :wq)
```

```
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks to Byte Commander on Stack Exchange)

# File_Name="Slab_TiN_111_TiN_Dimer_horizontal_N_terminated_automated_vasp_11x11x1"; mkdir ${File_Name}; cd ${File_Name}; touch ${File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt

#                         +-------+      +-------+
#                         |   N   |<--->|   Ti   |    Ti-N Dimer (1.6467A)
#                         +-------+      +-------+ |
#                                                  | Variable z-coord.
#                 +-------+              +-------+ v
#                 |   N   |<------------>|   N   |    TiN (111) slab, N-Terminated
#                 +-------+              +-------+

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-: Give the Following Variables -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Job_Name="Slab_TiN_111_TiN_Dimer_horizontal_N_terminated_automated_vasp_11x11x1" # Give the name you want to apply to all files here

Description="energies of automatically ranged and horizontally arranged TiN_Dimer bond lengths on a TiN (111) surface" # Please give a short description of the calculation for the README.txt file
Author="Steven E. Bopp, Materials Science & Engineering"

System_Name="Slab_TiN_111_TiN_Dimer_horizontal_N_terminated_11x11x1"     # Give a calculation title for VASP

Number_of_Elements="2"  # Give the number of individual elements that are in the simulation
Element_1="Ti"          # Give the symbol of the element 1 in your POSCAR file
Element_2="N"           # Give the symbol of the element 2 in your POSCAR file
Element_3="n/a"         # Give the symbol of the element 3 in your POSCAR file
Element_4="n/a"         # Give the symbol of the element 4 in your POSCAR file

Precision_Level="HIGH"   # VASP level of precision: NORMAL, MEDIUM, HIGH, LOW
KPOINTS_Grid="11 11 1"   # KGrid (give as wanted in the KPOINTS file e.g., '9 9 9')
Job_Time="02:01:00"      #Run time in hh:mm:ss
Job_Time_Min="00:39:59"  # Minimum run time of the script
Job_Nodes="1"            # Nodes that you want to use for the calculation
Job_Queue="flex"         # Calculation queue (e.g., 'debug', 'regular', 'flex', or 'low') low and flex queues have 50 and 75% discounts respectively for KNL
Architecture="knl"       # 'haswell' for Haswell chips or 'knl' for Knight's Landing)
Computer_Name="cluster"  # Give the name of the cluster
Module_Name="vasp"       # Module to load e.g., vasp, lammps, espresso, etc...
PP_Type="PBE"            # Pseudopotential type to load  e.g., PBE or LDA
Calculation_Type="full"  # Calculation type (e.g., 'test' will run 1 point and 'full' will run all of the points)
Submission_Type="1"      # 0 will NOT submit via sbatch and 1 WILL submit via sbatch
Piecewise_Submission="0" # 0 will NOT submit piecewise and 1 will submit piecewise, sleeping for 15 seconds between submissions)
Piecewise_Sleep="15"     # Give the number of seconds between job submission(s) for the Piecewise_Submission case (recommended to be ~30 to 50% of the run-time of a VASP calculation)
SQS_Update_Time="30"     # Give the number of seconds that you want the final squeue
```

```
 update cycle to wait between updates (recommended value: '30')

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:- BASH and bc Calculated Variables for VASP Automation :-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

case $Architecture in             # This switch/case will automatically switch the environments and executable locations for various architectures like haswell and knl
  haswell) echo " Haswell architecture selected: requires Intel PrgEnv"
           Processes_Per_Node="32" ;Threads_Per_Process="2"    # Recommended Haswell settings for 1 node and 32 MPI processes per node with 2 threads each
           Module_Location="~/uname/codes/vasp/vasp.5.3/vasp"         ;; # Executable location for HSW
  knl)     echo " Knight's Landing architecture selected
           Processes_Per_Node="64" ;Threads_Per_Process="4"    # Recommended MPI setting for 1 node. NPAR should be sqrt(Processes_Per_Node) !! NPAR = 8 gives substantial savings of ~67% compared to NPAR = 1
           Module_Location="~/uname/vasp_cray_compiled/vasp.5.3/vasp" ;; # Executable location for KNL
esac

case $Calculation_Type in         # This switch/case will automatically switch the calculation sequence based on the $Calculation_Type variable
  test)    echo " Running a test calculation"
           Sequence=$(seq -w 1 1 1); Number_of_Jobs=`echo $Sequence | wc -w`    ;; # Single point calculation sequence
  full)    echo " Running a full calculation"
           Sequence=$(seq -w 0.05 0.05 3.0); Number_of_Jobs=`echo $Sequence | wc -w`    ;; # Multiple point calculation sequence
esac

case $PP_Type in                  # This switch/case will automatically switch the pseudopotential type based on the $PP_Type variable
  PBE)    echo " Using the VASP PBE.54 pseudopotentials"
           Pseudo_Location="~/uname/pseudopotentials/PBE/potpaw_PBE.54/" ;; # Location of pseudopotential files for POTCAR, select: LDA, PBE, PW91
  LDA)    echo " Using the VASP LDA pseudopotentials"
           Pseudo_Location="~/uname/pseudopotentials/LDA/potpaw_LDA/"    ;; # Location of pseudopotential files for POTCAR, select: LDA, PBE, PW91
esac

case $Job_Queue in                # This switch/case will automatically insert options into the sbatch file
  flex)    Time_Min="#SBATCH --time-min=${Job_Time_Min}" ;; # Inserts #SBATCH --time-min=0:30:00 into sbatch
  shared)  Shared="#SBATCH --shared"                      ;; # Inserts #SBATCH --shared into sbatch
esac

Date=$(date '+%d/%m/%Y %H:%M:%S') # Give date in day/month/year hr/min/sec thanks user1293137 from https://unix.stackexchange.com/
NPAR=$(echo "scale=0;sqrt($Processes_Per_Node)" | bc) # Calculate NPAR to be inserted into INCAR file [~sqrt(job cores)]; uses BASH program bc; scale=0 sets bc to round to nearest integer (necessary for VASP)

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
```

```
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF
Job Name: ${Job_Name}.sh
This is a ${Module_Name} calculation of the ${System_Name} system to calculate ${
    Description} with ${PP_Type} pseudopotentials on a ${KPOINTS_Grid} grid..
Calculating with ${Number_of_Elements} element(s): ${Element_1}, ${Element_2}, ${
    Element_3}, and ${Element_4} for ${Job_Time} with ${Job_Nodes} job nodes on the
    ${Job_Queue} queue.
Using ${Architecture} node(s) with ${Processes_Per_Node} processes per node and ${
    Threads_Per_Process} threads per node for a maximum of ${Job_Time} hh:mm:ss.
Calculated with ${Computer_Name} by ${Author} on $Date.

A transcript of the calculation as seen from the terminal follows:

EOF

echo " Writing file README.txt..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Begin VASP File Creation :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:- Automatically Generate VASP Input Files -:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:- What Files Are Generated is Controlled by the Given Lattice Vars. -:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running a ${Module_Name} calculation of the ${System_Name} system to
    calculate ${Description} with ${PP_Type} pseudopotentials on a ${KPOINTS_Grid}
    grid."

echo " Calculating with ${Number_of_Elements} element(s): ${Element_1}, ${Element_2
    }, ${Element_3}, and ${Element_4} for ${Job_Time} with ${Job_Nodes} job nodes on
     the ${Job_Queue} queue."

echo " Using ${Architecture} node(s) with ${Processes_Per_Node} processes per node
    and ${Threads_Per_Process} threads per node for a maximum of ${Job_Time} hh:mm:
    ss"

echo " Calculated with ${Computer_Name} by ${Author} on $Date."

echo " Running script ${Job_Name}.sh..."

echo " The time is currently $Date "

echo " Automatically generating input files for ${Module_Name} "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:- Begin Main for() Loop for VASP File Creation :-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Current_Job_Number="1" # Begin a variable to use in the for() loop to count the
    current job up to the final job

for lat in ${Sequence}; do # Single calculation

```

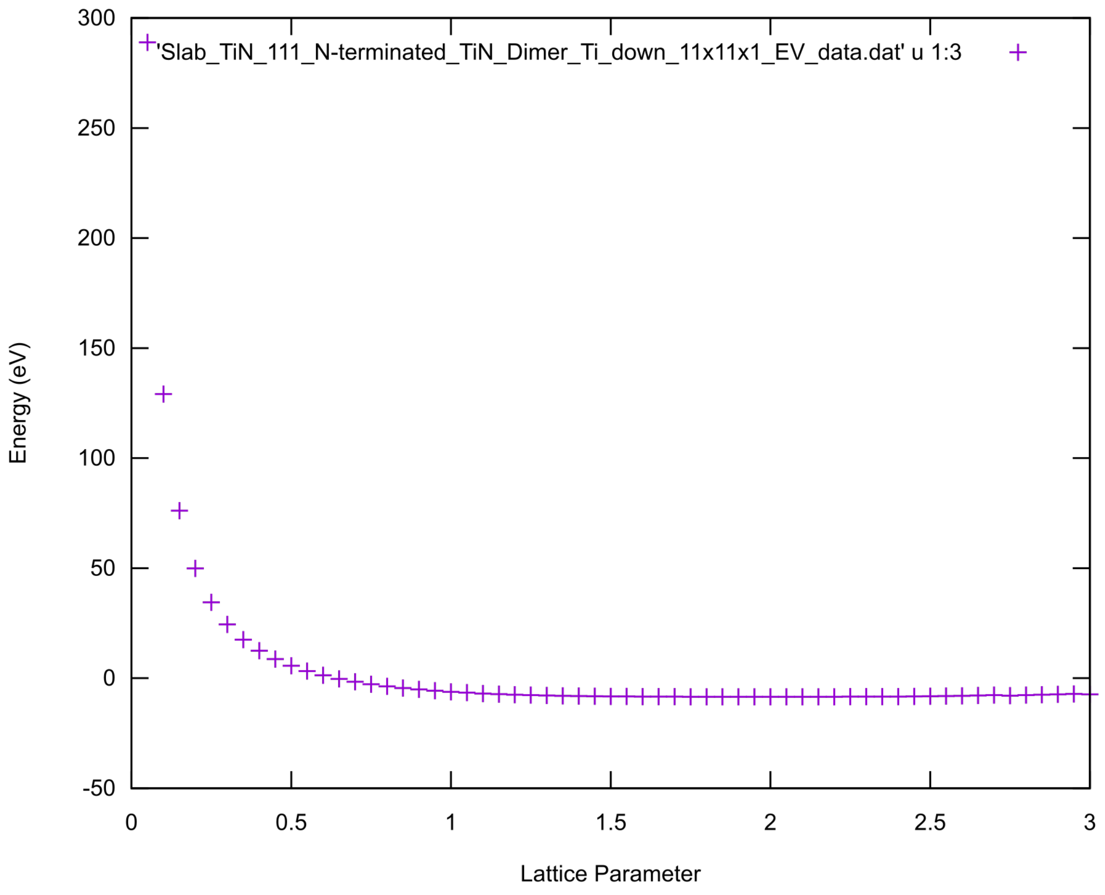
```
echo " The Lattice Constant Variable (lat) at this step = $lat"

mkdir ${System_Name}_a0_${lat}; cd ${System_Name}_a0_${lat} # Make and navigate into
     newly created directly for each iteration in the for() loop

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > ${Job_Name}_a0_${lat}.sb << EOF
#!/bin/bash
#SBATCH --job-name=${Job_Name}_a0_${lat}
#SBATCH -N ${Job_Nodes}
#SBATCH -C ${Architecture}
#SBATCH -q ${Job_Queue}
#SBATCH -t ${Job_Time}
${Time_Min}
${Shared}

#OpenMP settings:
export OMP_NUM_THREADS=1
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

module load ${Module_Name}
srun -n ${Processes_Per_Node} -c ${Threads_Per_Process} --cpu_bind=cores ${
    Module_Location}

EOF

echo " Writing input file ${Job_Name}_a0_${lat}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:- Create POSCAR File for VASP -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Top_Atom="3.66761759"
Titanium_Nitride_Dimer_Bond_Length="1.6467"

lat_a=$(echo "scale=9; $Top_Atom + $lat" | bc)

cat > POSCAR << !
# Rocksalt TiN oriented X=[11-2] Y=[1-10] Z=[111].
1.000000
       5.18679453         0.00000000         0.00000000
       0.00000000         5.98919444         0.00000000
       0.00000000         0.00000000        15.00000000
 Ti   N
 9    9
Cartesian
       0.00000000         0.00000000         0.00000000
       1.72893151         0.00000000         2.44507839
       4.32232878         1.49729861         2.44507839
       2.59339727         1.49729861         0.00000000
       0.00000000         2.99459722         0.00000000
       1.72893151         2.99459722         2.44507839
       4.32232878         4.49189583         2.44507839
```

```
      2.59339727        4.49189583        0.00000000
      2.59339727        1.49729861        ${lat_a}
      2.59339727        3.14399861        ${lat_a}
      0.86446576        1.49729861        1.22253920
      2.59339727        1.49729861        3.66761759
      3.45786302        0.00000000        1.22253920
      0.00000000        0.00000000        3.66761759
      0.86446576        4.49189583        1.22253920
      2.59339727        4.49189583        3.66761759
      3.45786302        2.99459722        1.22253920
      0.00000000        2.99459722        3.66761759
!

echo " Writing input file POSCAR..."
echo "  done"

# Dimer Equilibrium Bond Length
#    +-------+     +-------+
#    |Atom_1|<--->|Atom_2|  |
#    +-------+     +-------+ |  Variable z-coord. = lat_a
#                            |
#    +-----------------------+ v
#    |    Slab Surface      |
#    +-----------------------+

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create KPOINTS File for VASP :-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# ~ 1000 k-points per atom for metals will reduce error to approximately 10 meV; ~
    100 k-points per atom for insulators

cat > KPOINTS << EOF
Automatic mesh
0                    ! number of k-points = 0 -> automatic generation scheme
M
${KPOINTS_Grid}      ! subdivisions N_1, N_2 and N_3 along recipr. latt. vectors
0  0  0              ! optional shift of the mesh (s_1, s_2, s_3)
EOF

echo " Writing input file KPOINTS..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Create INCAR File for VASP :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# This INCAR is specific for semiconductors and insulators!

cat > INCAR << EOF
# General Setup
  System = ${System_Name}      # Calculation Title
  PREC   = ${Precision_Level} # Options: Normal, Medium, High, Low

  ISMEAR = 1        # method of Methfessel-Paxton order N
  SIGMA  = 0.1      # specifies the width of the smearing in eV
  ENCUT  = 425      # specifies the cutoff energy for the plane-wave-basis set in eV
  ALGO   = FAST     # mixture of the Davidson and RMM-DIIS algorithms
```

```

# Parallelization
  NPAR    = ${NPAR}         # approx. SQRT(number of cores)
EOF

echo " Writing input file INCAR..."
echo "  done"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-: Create file POTCAR -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# This switch/case will automatically generate a POTCAR file based on the options in the header of this script
# Create POTCAR with cat in the exact order that elements appear in the POSCAR file

case $Number_of_Elements in

  1) echo " One element selected for simulation: $Element_1"
     cat ${Pseudo_Location}${Element_1}/POTCAR > POTCAR ;;
  2) echo " Two elements selected for simulation: $Element_1 $Element_2"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR > POTCAR ;;
  3) echo " Three elements selected for simulation: $Element_1 $Element_2 $Element_3"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR ${Pseudo_Location}${Element_3}/POTCAR > POTCAR ;;
  4) echo " Four elements selected for simulation: $Element_1 $Element_2 $Element_3 $Element_4"
     cat ${Pseudo_Location}${Element_1}/POTCAR ${Pseudo_Location}${Element_2}/POTCAR ${Pseudo_Location}${Element_3}/POTCAR ${Pseudo_Location}${Element_4}/POTCAR > POTCAR ;;

esac

echo " Writing input file POTCAR..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:- Run VASP Calculation with SBATCH -:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Submitting ${Job_Name}_a0_${lat}.sb via sbatch..."

echo " Running job ${Current_Job_Number} out of ${Number_of_Jobs} job(s) on the ${Job_Queue} queue with ${Job_Nodes} node(s) per job for ${Job_Time} each"

case $Submission_Type in # This switch/case will automatically switch whether to or not to submit job(s) to the queue
  0)    echo " Not submitting job(s)"   ;; # Not submitting jobs to queue
  1)    sbatch ${Job_Name}_a0_${lat}.sb ;; # Submit job to queue
esac

Current_Job_Number=$(echo `expr $Current_Job_Number + 1`)  # Add 1 to the Current_Job_Number counter variable

case $Piecewise_Submission in                    # This switch/case will
```

```
    automatically switch whether to or not to submit job(s) to the queue
  0)                                  ;; # Not submitting jobs to queue
  1)    squeue -u uname | head; sleep ${Piecewise_Sleep} ;; # Submit job to queue if
     trying to run batches for debugging on debug queue
esac

echo "  Begin:"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-: End VASP File Creation -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-: End Main for() Loop for VASP File Creation -:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cd .. # Navigate out of newly created directory

done  # End of main for() loop

#echo " The contents of this directory are now the following:" ; ls # List newly
    created directories

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:End Elapsed Time Measurement-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

squeue -u uname

echo " The time is currently $Date "

echo " Dumping terminal session into README.txt"

echo " Success, End of Script, Running squeue -u uname on a 5 Second Loop"

while [ 1 ] ; do Jobs=$(squeue -u uname | wc -l); echo "`expr ${Jobs} - 1` jobs in
    the queue"; squeue -u uname; date; sleep ${SQS_Update_Time}; done # continue to
    update the squeue every 5 seconds
```

Using Gnuplot (for more information, see section 3.8), we can visualize what the results of a sample calculation for this system would be as follows:



And, like in other sections, using some mathematical analysis in MATLAB, we can also determine what the absolute minimum energy value in the calculation set which corresponds to the equilibrium configuration. In section 9.8 we will see how to automatically extract all of the pertinent data from these calculations, suffice it to say here however that we have this data already collected and will import it into MATLAB for analysis. Please see the following .dat file from which the above plot was created:

```
"Lattice Parameter a0" "Total Energy E0" "Energy Per Ion" "Volume Per Ion in A"
0.05 0.52006751E+04 288.926 25.89
0.10 0.23243415E+04 129.13 25.89
0.15 0.13703399E+04 76.13 25.89
0.20 0.89852067E+03 49.9178 25.89
0.25 0.62055720E+03 34.4754 25.89
0.30 0.43998116E+03 24.4434 25.89
0.35 0.31504570E+03 17.5025 25.89
0.40 0.22443393E+03 12.4686 25.89
0.45 0.15586896E+03 8.65939 25.89
0.50 0.10213118E+03 5.67395 25.89
0.55 0.59017189E+02 3.27873 25.89
0.60 0.23908345E+02 1.32824 25.89
```

```
0.65 −.49428874E+01 −0.274605 25.89
0.70 −.28844958E+02 −1.6025 25.89
0.75 −.48799955E+02 −2.71111 25.89
0.80 −.65609699E+02 −3.64498 25.89
0.85 −.79848490E+02 −4.43603 25.89
0.90 −.91949856E+02 −5.10833 25.89
0.95 −.10224292E+03 −5.68016 25.89
1.00 −.11100109E+03 −6.16673 25.89
1.05 −.11846209E+03 −6.58123 25.89
1.10 −.12482409E+03 −6.93467 25.89
1.15 −.13023600E+03 −7.23533 25.89
1.20 −.13480058E+03 −7.48892 25.89
1.25 −.13859434E+03 −7.69969 25.89
1.30 −.14169456E+03 −7.87192 25.89
1.35 −.14417901E+03 −8.00995 25.89
1.40 −.14613653E+03 −8.1187 25.89
1.45 −.14765363E+03 −8.20298 25.89
1.50 −.14881189E+03 −8.26733 25.89
1.55 −.14968276E+03 −8.31571 25.89
1.60 −.15032905E+03 −8.35161 25.89
1.65 −.15080045E+03 −8.3778 25.89
1.70 −.15113533E+03 −8.39641 25.89
1.75 −.15136489E+03 −8.40916 25.89
1.80 −.15151354E+03 −8.41742 25.89
1.85 −.15159954E+03 −8.4222 25.89
1.90 −.15163737E+03 −8.4243 25.89
1.95 −.15163834E+03 −8.42435 25.89
2.00 −.15161094E+03 −8.42283 25.89
2.05 −.15156141E+03 −8.42008 25.89
2.10 −.15149510E+03 −8.41639 25.89
2.15 −.15141609E+03 −8.41201 25.89
2.20 −.15132742E+03 −8.40708 25.89
2.25 −.15123176E+03 −8.40176 25.89
2.30 −.15113038E+03 −8.39613 25.89
2.35 −.15102515E+03 −8.39029 25.89
2.40 −.15090103E+03 −8.38339 25.89
2.45 −.14935582E+03 −8.29755 25.89
2.50 −.14764123E+03 −8.20229 25.89
2.55 −.14627167E+03 −8.1262 25.89
2.60 −.14356170E+03 −7.97565 25.89
2.65 −.14113231E+03 −7.84068 25.89
2.70 −.13868309E+03 −7.70462 25.89
2.75 −.14196237E+03 −7.8868 25.89
2.80 −.13859814E+03 −7.6999 25.89
2.85 −.13527883E+03 −7.51549 25.89
2.90 −.13183631E+03 −7.32424 25.89
2.95 −.12797128E+03 −7.10952 25.89
3.00 −.13160835E+03 −7.31158 25.89
```

Importing this data (the first and third columns) into MATLAB, we can use splines and the 'find min' functionality to find the minimum energy configuration based on interpolation:

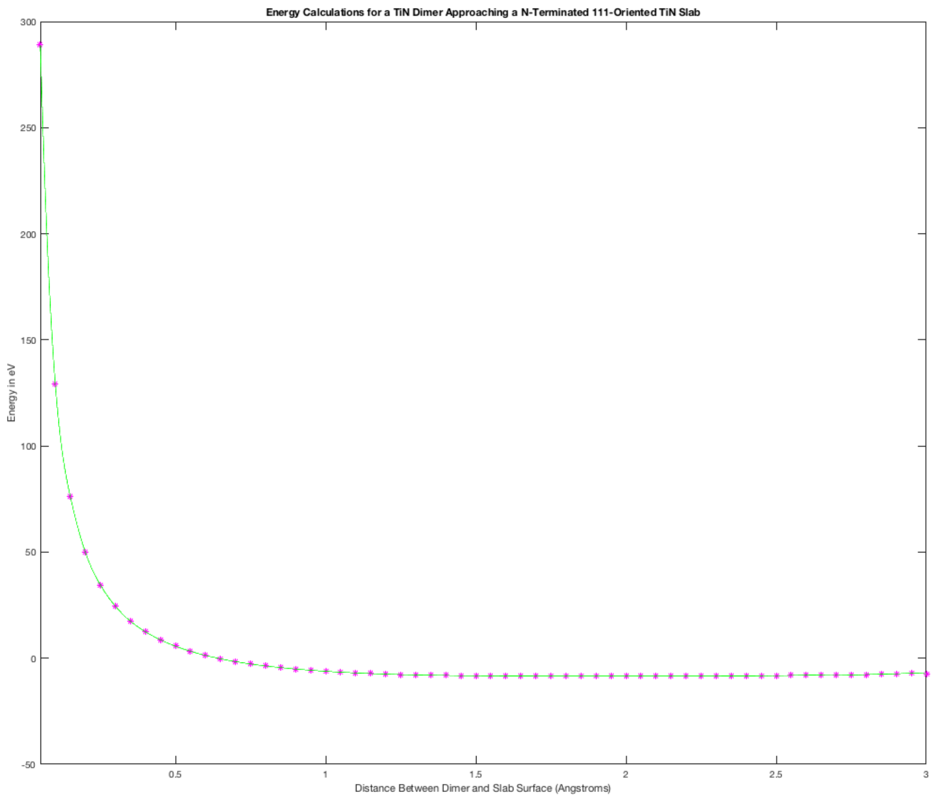
```
%% TiN_Dimer_Approaching_TiN.m
%  Written by Steven E. Bopp

%%
Lattice_Parameter=[0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
1.05
1.10
1.15
1.20
1.25
1.30
1.35
1.40
1.45
1.50
1.55
1.60
1.65
1.70
1.75
1.80
1.85
1.90
1.95
2.00
2.05
2.10
2.15
2.20
2.25
2.30
2.35
2.40
2.45
2.50
2.55
2.60
2.65
2.70
2.75
2.80
2.85
2.90
2.95
3.00];

```

```
Energy_Per_Ion=[ 271.272
 119.077
 68.7155
 43.9278
 29.4373
 20.1302
 13.8004
 9.33605
 6.11001
 3.73753
 1.96417
 0.611231
 −0.45174
 −1.32361
 −2.07586
 −2.75038
 −3.35932
 −3.90081
 −4.37184
 −4.77362
 −5.11125
 −5.39189
 −5.62318
 −5.81244
 −5.96631
 −6.09071
 −6.19078
 −6.27102
 −6.33525
 −6.38672
 −6.4281
 −6.46152
 −6.48874
 −6.51108
 −6.52962
 −6.54519
 −6.55841
 −6.56974
 −6.57952
 −6.58804
 −6.59548
 −6.60201
 −6.60774
 −6.61278
 −6.61719
 −6.62106
 −6.62442
 −6.62733
 −6.62984
 −6.63197
 −6.63381
 −6.63533
 −6.63654
 −6.63753
 −6.63823
 −6.6387
 −6.63894
 −6.63898
 -6.63882
 -6.63849];

%%

E0=Energy_Per_Ion';
x=Lattice_Parameter';  xx=[0.05:.0001:3.00];
yy=spline(x,E0,xx);         % Create spline interpolate for the lattice constant

figure(1);
plot(x,E0,'m*',xx,yy,'g'); % Plot spline interpolate and Energy with Lattice
    Parameter
xlabel('Distance Between Dimer and Slab Surface (Angstroms)'); ylabel('Energy in eV'
    );
title('Energy Calculations for a TiN Dimer Approaching a N-Terminated 111-Oriented
    TiN Slab');
xlim([0.05 3.00])
% Pentagram p; hexagram h; diamond d; square s;

indexmin=find(min(yy) == yy);                 % Define indexmin
xmin = xx(indexmin); ymin = yy(indexmin);   % Calculate minimum values
A0 = xmin;
fprintf('Minimum Energy Configuration Length:%g \n',A0) % Display lattice constant
    and error from expected value
```

MATLAB should report the following:

```
Minimum Energy Configuration Length:1.9258
```

MATLAB should also generate the following plot of what it has fit with the spline interpolation:



What this specifically means is that, from the spline interpolation, MATLAB has calculated that a distance of 1.9258 angstroms for this atomic configuration of a horizontal TiN dimer approaching the 111 surface of a TiN slab is the minimum energy configuration and therefore corresponds to where a bond will form.

9.8 A script to collect and collate energy and volume parameters from VASP calculations and notify the user for unconverged calculations with for loops, awk, tail, and tput bel

As we will talk about in sections 9.9, and 9.7, it is often useful to run many VASP calculations with slightly varied parameters. This can be done for a variety of reasons but one example is to determine the bulk modulus from fitting with the Birch–Murnaghan equation of state. You can use information like the ground state energy per ion and the volume per ion to determine many optimal conditions for a crystal or atomic system such as minimum energy configurations that might correspond to an equilibrium crystal structure for example.

However, like a discussion of vasprun.xml files that will follow in section 9.9, the files containing this information can be spread between many directories and contained within different files within those directories. Additionally, it is not always apparent whether a calculation within a series of calculations has been converged unless each of these files are either inspected by hand or checked with an automated script. The shell is a great tool for extracting and collating all of that information. One such means of doing so is with the script that is attached in this section.

This script is separated into four major sections. The first of these sections starts at the comment "Create Data Storage File" and ends at the comment "Begin Main for() Loop for VASP Data Collection". Section one creates naming variables by automatically extracting the system name(s) from a series of VASP calculation output files so that the user does not need to manually add this input and many series of calculations may be analyzed or extracted simultaneously. Awk is used to search the input shell script used to generate the calculations and then extract the system name. The extracted system name is then modified slightly with the cut and reverse (rev) commands to remove several characters which were unwanted. Finally in section one, we create an empty data file where the extracted vasp calculation data will be stored.

Section two of this script picks up where section one left off and continues up to the comment "Echo terminal commands to create a datfile and plots in OS X". This section of the script uses a for loop to do its heavy lifting. The for loop creates a variable named *a* that operates over every sub-directory within the current directory. First, the loop checks to see whether the calculation it is currently considering has converged by checking the OSCIZAR file by using awk to search for the line where VASP reports "F=" followed by a text field. If that following text field is "1", then the calculation has converged and the word 'converged' will be printed to the terminal. In the case that the following text field is anything other than "1", then the terminal uses the command tput bel to ring a bell noise and print '<!> Warning: Calculation Did Not Converge <!>' to the terminal. I find this to be extremely useful because you can scan thousands (or an arbitrary number) of calculations for whether they have converged within the time limits that you have given to your cluster for each calculation. This is useful when you are attempting to request a minimum amount of time necessary to converge a specific variety of calculation so that calculations will start running on supercomputer queues earlier rather than later because frequently calculations requesting a smaller amount of computational resources will be submitted to computers earlier than those requesting a large amount of resources.

Continuing in section two, we use awk heavily to extract (and store as variables) the number of ions, the total energy E0, the energy per ion, and the volume per ion from the OUTCAR and OSZICAR files. The lattice parameter at the current step is also stored as a variable at this step. Completing the loop, we navigate up one directory and then append the variables that we have just stored into the data file created earlier. The loop continues until all of the sub-directories have been searched, exhausting the initial conditions.

Section three continues where section two has left off and ends at the comment "Create data plotter program". This section of the script was made to ease the transition from a remote machine to a local machine by echoing lots of information to the terminal (so that copy and paste can be used instead of scp or some other protocol) as well as specific BASH commands. Overall, section three is intended to concatenate all of the information to the shell that we have just collected in the format of a series of BASH commands which can just be copied and pasted into a local terminal that will automatically generate those same files on your local system. For the Mac OS users, there is a special treat built into this section too in the form of a qlmanage command that generates a .png format image from a Gnuplot .svg image which I find very nice for inserting into research group meeting slides without having to fuss over converting the images from one format to another by hand. Again, as said many times before, automating things in this manner makes all of these operations much easier.

Finally, section four encompasses everything after the comment "Create data plotter program". This final section of the script creates a Gnuplot script and then executes it to plot all of the relevant data that we have just collected to the terminal for quick viewing. Plotting with the 'dumb' terminal in Gnuplot (see section 3.8 for more on Gnuplot) allows us to plot headless in the terminal and make sure that the general behavior of a series of data are at least reasonably aligned with our expectations.

Please see the following script to collect and collate energy and volume information from many VASP calculations and an example of its use on a remote cluster:

```
#!/bin/bash

# The purpose of this code is to collect all of the energy and volume data from a
    series of directories
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks
     to Byte Commander on Stack Exchange)

# File_Name="VASP_EV_Collector_v3"; touch ${File_Name}.sh; chmod +x ${File_Name}.sh;
     vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README_EV_Collector.txt

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Create Data Storage File :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

System_Name=$(awk '/System_Name=/{print $1}' *vasp.sh | cut -c14- | rev | cut -c2- |
     rev)

File_Name=${System_Name}_EV_data.dat

cat > ${File_Name} << EOF
"Lattice Parameter a0" "Total Energy E0" "Energy Per Ion" "Volume Per Ion in A"
EOF

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-: Begin Main for() Loop for VASP Data Collection -:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

for a in */ ; do # Loop within a single calculation directory
  echo "  Entering directory $a"
  cd "$a" #; pwd

    Convergence=$(awk '/F=/{print $1}' OSZICAR)
    if [[ $Convergence = 1 ]] ; then echo '  Converged' ; else tput bel && echo '<!>
     Warning: Calculation Did Not Converge <!>' ; fi

    Number_of_Ions=$(awk '/NIONS/{print $12}' OUTCAR) # Collect the number of ions
    from the OSZICAR file
    tail -1 OSZICAR # Print the last line of OSZICAR to the terminal, make sure that
     the program terminated normally and converged
    E0=$(tail -n1 OSZICAR | awk '{print $5}') # Collect E0 from the final SCF
    iteration
    Energy_Per_Ion=$(tail -n1 OSZICAR | awk -v Number_of_Ions="$Number_of_Ions" '{
    print $5/Number_of_Ions}') # Collect E0 from the final SCF iteration and divide
    by number of ions
    Volume_Per_Ion=$(awk '/ion in A,a.u./{print $5}' OUTCAR) # Collect the volume
    per ion by searching for a truncated part of the string 'volume/ion in A,a.u.'

    a0=$(echo $a | sed 's/.*_a0_//' | rev | cut -c2- | rev)
    # Extracts the variable $a into sed which transforms (for example) Al2O3_a0_0
    .980000000 into (for example) 0.980000000 -> stored as $a0
    # This command needs the rev and cut functions because it stores the a0 as
    something like 0.980000000/ and we need to remove the errant / as well

    echo "  For lattice parameter = ${a0}, Total energy = ${E0} Energy per ion = ${
    Energy_Per_Ion} Volume per ion = ${Volume_Per_Ion} "

  cd ..
  echo $a0 $E0 $Energy_Per_Ion $Volume_Per_Ion >> ${File_Name}

done # End of main for() loop

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-: Echo terminal commands to create a datfile and plots in OS X -:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

SVG_Name=${System_Name}_EV_data.svg

echo ""; echo ""; echo ""
echo "cat > $File_Name << EOF"
cat $File_Name
echo "EOF"
echo ""
echo "gnuplot"
echo "set terminal svg enhanced"
echo "set xlabel 'Lattice Parameter'" # Multiplication Factor
echo "set ylabel 'Energy (eV)'"
echo "set out '$SVG_Name'; p '$File_Name' u 1:3"
echo "q"
echo ""
echo "qlmanage -t -s 1000 -o . $SVG_Name" # Convert an svg to a png image on osx
    thanks to superuser.com user tst
echo ""
echo "clear"
echo ""; echo ""; echo ""

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Create data plotter program :-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > EV_Plotter.sh << "EOF"
#!/bin/bash

Dat_File=$(ls *.dat)

## Plot Data with GNUPlot
gnuplot <<!
set terminal dumb size 105,55
set xlabel "Lattice Parameter " # Multiplication Factor
set ylabel "Energy (eV)"
p "$Dat_File" u 1:3 with lines
!
EOF

```

```
echo ""
echo " Writing file EV_Plotter.sh..."; echo "  done"
chmod +x EV_Plotter.sh

./EV_Plotter.sh

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-: End of directions, halt :-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " VASP data collection and compilation completed"
echo " Success, End of Script"
```

The output of this script can be seen below for a series of calculations based on the script presented in section 9.7 which concerns the adsorption of a TiN dimer horizontally on the 111 surface of a TiN slab. Note that all of the calculations converged properly, if they had not then the terminal would have made a bell sound and printed '<!> Warning: Calculation Did Not Converge <!>' to the terminal before continuing to the next step in the for loop.

```
sbopp@cori08:/global/cscratch1/sd/sbopp/vasp_meam_calculations_PBE/Slabs_Ti-N/
    Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_automated_vasp_11x11x1> cat
    README_EV_Collector.txt
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.05/
  Converged
   1 F= 0.52006751E+04 E0= 0.52006751E+04  d E =0.000000E+00
  For lattice parameter = 0.05, Total energy = 0.52006751E+04 Energy per ion =
    288.926 Volume per ion = 25.89
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.10/
  Converged
   1 F= 0.23243415E+04 E0= 0.23243415E+04  d E =0.000000E+00
  For lattice parameter = 0.10, Total energy = 0.23243415E+04 Energy per ion =
    129.13 Volume per ion = 25.89
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.15/
  Converged
   1 F= 0.13703399E+04 E0= 0.13703399E+04  d E =0.000000E+00
  For lattice parameter = 0.15, Total energy = 0.13703399E+04 Energy per ion = 76.13
     Volume per ion = 25.89
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.20/
  Converged
   1 F= 0.89852067E+03 E0= 0.89852067E+03  d E =0.000000E+00
  For lattice parameter = 0.20, Total energy = 0.89852067E+03 Energy per ion =
    49.9178 Volume per ion = 25.89
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.25/
  Converged
   1 F= 0.62055720E+03 E0= 0.62055720E+03  d E =0.000000E+00
  For lattice parameter = 0.25, Total energy = 0.62055720E+03 Energy per ion =
    34.4754 Volume per ion = 25.89
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.30/
  Converged
   1 F= 0.43998116E+03 E0= 0.43998116E+03  d E =0.000000E+00
  For lattice parameter = 0.30, Total energy = 0.43998116E+03 Energy per ion =
    24.4434 Volume per ion = 25.89
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.35/
  Converged
   1 F= 0.31504570E+03 E0= 0.31504570E+03  d E =0.000000E+00
  For lattice parameter = 0.35, Total energy = 0.31504570E+03 Energy per ion =
    17.5025 Volume per ion = 25.89
  Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.40/
```

```
Converged
 1 F= 0.22443393E+03 E0= 0.22443393E+03  d E =0.000000E+00
For lattice parameter = 0.40, Total energy = 0.22443393E+03 Energy per ion =
  12.4686 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.45/
Converged
 1 F= 0.15586896E+03 E0= 0.15586896E+03  d E =0.000000E+00
For lattice parameter = 0.45, Total energy = 0.15586896E+03 Energy per ion =
  8.65939 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.50/
Converged
 1 F= 0.10213118E+03 E0= 0.10213118E+03  d E =0.000000E+00
For lattice parameter = 0.50, Total energy = 0.10213118E+03 Energy per ion =
  5.67395 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.55/
Converged
 1 F= 0.59017189E+02 E0= 0.59017189E+02  d E =0.000000E+00
For lattice parameter = 0.55, Total energy = 0.59017189E+02 Energy per ion =
  3.27873 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.60/
Converged
 1 F= 0.23908345E+02 E0= 0.23908345E+02  d E =0.000000E+00
For lattice parameter = 0.60, Total energy = 0.23908345E+02 Energy per ion =
  1.32824 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.65/
Converged
 1 F= -.49428874E+01 E0= -.49428874E+01  d E =0.000000E+00
For lattice parameter = 0.65, Total energy = -.49428874E+01 Energy per ion =
  -0.274605 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.70/
Converged
 1 F= -.28844958E+02 E0= -.28844958E+02  d E =0.000000E+00
For lattice parameter = 0.70, Total energy = -.28844958E+02 Energy per ion =
  -1.6025 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.75/
Converged
 1 F= -.48799955E+02 E0= -.48799955E+02  d E =0.000000E+00
For lattice parameter = 0.75, Total energy = -.48799955E+02 Energy per ion =
  -2.71111 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.80/
Converged
 1 F= -.65609699E+02 E0= -.65609699E+02  d E =0.000000E+00
For lattice parameter = 0.80, Total energy = -.65609699E+02 Energy per ion =
  -3.64498 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.85/
Converged
 1 F= -.79848490E+02 E0= -.79848490E+02  d E =0.000000E+00
For lattice parameter = 0.85, Total energy = -.79848490E+02 Energy per ion =
  -4.43603 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.90/
Converged
 1 F= -.91949856E+02 E0= -.91949856E+02  d E =0.000000E+00
For lattice parameter = 0.90, Total energy = -.91949856E+02 Energy per ion =
  -5.10833 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_0.95/
Converged
 1 F= -.10224292E+03 E0= -.10224292E+03  d E =0.000000E+00
For lattice parameter = 0.95, Total energy = -.10224292E+03 Energy per ion =
  −5.68016 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.00/
Converged
 1 F= −.11100109E+03 E0= −.11100109E+03  d E =0.000000E+00
For lattice parameter = 1.00, Total energy = −.11100109E+03 Energy per ion =
  −6.16673 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.05/
Converged
 1 F= −.11846209E+03 E0= −.11846209E+03  d E =0.000000E+00
For lattice parameter = 1.05, Total energy = −.11846209E+03 Energy per ion =
  −6.58123 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.10/
Converged
 1 F= −.12482409E+03 E0= −.12482409E+03  d E =0.000000E+00
For lattice parameter = 1.10, Total energy = −.12482409E+03 Energy per ion =
  −6.93467 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.15/
Converged
 1 F= −.13023600E+03 E0= −.13023600E+03  d E =0.000000E+00
For lattice parameter = 1.15, Total energy = −.13023600E+03 Energy per ion =
  −7.23533 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.20/
Converged
 1 F= −.13480058E+03 E0= −.13480058E+03  d E =0.000000E+00
For lattice parameter = 1.20, Total energy = −.13480058E+03 Energy per ion =
  −7.48892 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.25/
Converged
 1 F= −.13859434E+03 E0= −.13859434E+03  d E =0.000000E+00
For lattice parameter = 1.25, Total energy = −.13859434E+03 Energy per ion =
  −7.69969 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.30/
Converged
 1 F= −.14169456E+03 E0= −.14169456E+03  d E =0.000000E+00
For lattice parameter = 1.30, Total energy = −.14169456E+03 Energy per ion =
  −7.87192 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.35/
Converged
 1 F= −.14417901E+03 E0= −.14417901E+03  d E =0.000000E+00
For lattice parameter = 1.35, Total energy = −.14417901E+03 Energy per ion =
  −8.00995 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.40/
Converged
 1 F= −.14613653E+03 E0= −.14613653E+03  d E =0.000000E+00
For lattice parameter = 1.40, Total energy = −.14613653E+03 Energy per ion =
  −8.1187 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.45/
Converged
 1 F= −.14765363E+03 E0= −.14765363E+03  d E =0.000000E+00
For lattice parameter = 1.45, Total energy = −.14765363E+03 Energy per ion =
  −8.20298 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.50/
Converged
 1 F= −.14881189E+03 E0= −.14881189E+03  d E =0.000000E+00
For lattice parameter = 1.50, Total energy = −.14881189E+03 Energy per ion =
  −8.26733 Volume per ion = 25.89
Entering directory Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.55/
Converged
```

```
  1 F= -.14968276E+03 E0= -.14968276E+03  d E =0.000000E+00
 For lattice parameter = 1.55, Total energy = -.14968276E+03 Energy per ion =
   -8.31571 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.60/
 Converged
  1 F= -.15032905E+03 E0= -.15032905E+03  d E =0.000000E+00
 For lattice parameter = 1.60, Total energy = -.15032905E+03 Energy per ion =
   -8.35161 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.65/
 Converged
  1 F= -.15080045E+03 E0= -.15080045E+03  d E =0.000000E+00
 For lattice parameter = 1.65, Total energy = -.15080045E+03 Energy per ion =
   -8.3778 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.70/
 Converged
  1 F= -.15113533E+03 E0= -.15113533E+03  d E =0.000000E+00
 For lattice parameter = 1.70, Total energy = -.15113533E+03 Energy per ion =
   -8.39641 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.75/
 Converged
  1 F= -.15136489E+03 E0= -.15136489E+03  d E =0.000000E+00
 For lattice parameter = 1.75, Total energy = -.15136489E+03 Energy per ion =
   -8.40916 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.80/
 Converged
  1 F= -.15151354E+03 E0= -.15151354E+03  d E =0.000000E+00
 For lattice parameter = 1.80, Total energy = -.15151354E+03 Energy per ion =
   -8.41742 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.85/
 Converged
  1 F= -.15159954E+03 E0= -.15159954E+03  d E =0.000000E+00
 For lattice parameter = 1.85, Total energy = -.15159954E+03 Energy per ion =
   -8.4222 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.90/
 Converged
  1 F= -.15163737E+03 E0= -.15163737E+03  d E =0.000000E+00
 For lattice parameter = 1.90, Total energy = -.15163737E+03 Energy per ion =
   -8.4243 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_1.95/
 Converged
  1 F= -.15163834E+03 E0= -.15163834E+03  d E =0.000000E+00
 For lattice parameter = 1.95, Total energy = -.15163834E+03 Energy per ion =
   -8.42435 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.00/
 Converged
  1 F= -.15161094E+03 E0= -.15161094E+03  d E =0.000000E+00
 For lattice parameter = 2.00, Total energy = -.15161094E+03 Energy per ion =
   -8.42283 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.05/
 Converged
  1 F= -.15156141E+03 E0= -.15156141E+03  d E =0.000000E+00
 For lattice parameter = 2.05, Total energy = -.15156141E+03 Energy per ion =
   -8.42008 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.10/
 Converged
  1 F= -.15149510E+03 E0= -.15149510E+03  d E =0.000000E+00
 For lattice parameter = 2.10, Total energy = -.15149510E+03 Energy per ion =
   -8.41639 Volume per ion = 25.89
```

```
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.15/
Converged
 1 F= -.15141609E+03 E0= -.15141609E+03  d E =0.000000E+00
For lattice parameter = 2.15, Total energy = -.15141609E+03 Energy per ion =
  -8.41201 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.20/
Converged
 1 F= -.15132742E+03 E0= -.15132742E+03  d E =0.000000E+00
For lattice parameter = 2.20, Total energy = -.15132742E+03 Energy per ion =
  -8.40708 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.25/
Converged
 1 F= -.15123176E+03 E0= -.15123176E+03  d E =0.000000E+00
For lattice parameter = 2.25, Total energy = -.15123176E+03 Energy per ion =
  -8.40176 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.30/
Converged
 1 F= -.15113038E+03 E0= -.15113038E+03  d E =0.000000E+00
For lattice parameter = 2.30, Total energy = -.15113038E+03 Energy per ion =
  -8.39613 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.35/
Converged
 1 F= -.15102515E+03 E0= -.15102515E+03  d E =0.000000E+00
For lattice parameter = 2.35, Total energy = -.15102515E+03 Energy per ion =
  -8.39029 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.40/
Converged
 1 F= -.15090103E+03 E0= -.15090103E+03  d E =0.000000E+00
For lattice parameter = 2.40, Total energy = -.15090103E+03 Energy per ion =
  -8.38339 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.45/
Converged
 1 F= -.14935582E+03 E0= -.14935582E+03  d E =0.000000E+00
For lattice parameter = 2.45, Total energy = -.14935582E+03 Energy per ion =
  -8.29755 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.50/
Converged
 1 F= -.14764123E+03 E0= -.14764123E+03  d E =0.000000E+00
For lattice parameter = 2.50, Total energy = -.14764123E+03 Energy per ion =
  -8.20229 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.55/
Converged
 1 F= -.14627167E+03 E0= -.14627167E+03  d E =0.000000E+00
For lattice parameter = 2.55, Total energy = -.14627167E+03 Energy per ion =
  -8.1262 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.60/
Converged
 1 F= -.14356170E+03 E0= -.14356170E+03  d E =0.000000E+00
For lattice parameter = 2.60, Total energy = -.14356170E+03 Energy per ion =
  -7.97565 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.65/
Converged
 1 F= -.14113231E+03 E0= -.14113231E+03  d E =0.000000E+00
For lattice parameter = 2.65, Total energy = -.14113231E+03 Energy per ion =
  -7.84068 Volume per ion = 25.89
Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.70/
Converged
 1 F= -.13868309E+03 E0= -.13868309E+03  d E =0.000000E+00
 For lattice parameter = 2.70, Total energy = -.13868309E+03 Energy per ion =
   -7.70462 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.75/
 Converged
  1 F= -.14196237E+03 E0= -.14196237E+03  d E =0.000000E+00
 For lattice parameter = 2.75, Total energy = -.14196237E+03 Energy per ion =
   -7.8868 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.80/
 Converged
  1 F= -.13859814E+03 E0= -.13859814E+03  d E =0.000000E+00
 For lattice parameter = 2.80, Total energy = -.13859814E+03 Energy per ion =
   -7.6999 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.85/
 Converged
  1 F= -.13527883E+03 E0= -.13527883E+03  d E =0.000000E+00
 For lattice parameter = 2.85, Total energy = -.13527883E+03 Energy per ion =
   -7.51549 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.90/
 Converged
  1 F= -.13183631E+03 E0= -.13183631E+03  d E =0.000000E+00
 For lattice parameter = 2.90, Total energy = -.13183631E+03 Energy per ion =
   -7.32424 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_2.95/
 Converged
  1 F= -.12797128E+03 E0= -.12797128E+03  d E =0.000000E+00
 For lattice parameter = 2.95, Total energy = -.12797128E+03 Energy per ion =
   -7.10952 Volume per ion = 25.89
 Entering directory Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_a0_3.00/
 Converged
  1 F= -.13160835E+03 E0= -.13160835E+03  d E =0.000000E+00
 For lattice parameter = 3.00, Total energy = -.13160835E+03 Energy per ion =
   -7.31158 Volume per ion = 25.89



cat > Slab_TiN_111_N-terminated_TiN_Dimer_Ti_down_11x11x1_EV_data.dat << EOF
"Lattice Parameter a0" "Total Energy E0" "Energy Per Ion" "Volume Per Ion in A"
0.05 0.52006751E+04 288.926 25.89
0.10 0.23243415E+04 129.13 25.89
0.15 0.13703399E+04 76.13 25.89
0.20 0.89852067E+03 49.9178 25.89
0.25 0.62055720E+03 34.4754 25.89
0.30 0.43998116E+03 24.4434 25.89
0.35 0.31504570E+03 17.5025 25.89
0.40 0.22443393E+03 12.4686 25.89
0.45 0.15586896E+03 8.65939 25.89
0.50 0.10213118E+03 5.67395 25.89
0.55 0.59017189E+02 3.27873 25.89
0.60 0.23908345E+02 1.32824 25.89
0.65 -.49428874E+01 -0.274605 25.89
0.70 -.28844958E+02 -1.6025 25.89
0.75 -.48799955E+02 -2.71111 25.89
0.80 -.65609699E+02 -3.64498 25.89
0.85 -.79848490E+02 -4.43603 25.89
0.90 -.91949856E+02 -5.10833 25.89
0.95 -.10224292E+03 -5.68016 25.89
1.00 -.11100109E+03 -6.16673 25.89
1.05 -.11846209E+03 -6.58123 25.89
```

```
1.10 −.12482409E+03 −6.93467 25.89
1.15 −.13023600E+03 −7.23533 25.89
1.20 −.13480058E+03 −7.48892 25.89
1.25 −.13859434E+03 −7.69969 25.89
1.30 −.14169456E+03 −7.87192 25.89
1.35 −.14417901E+03 −8.00995 25.89
1.40 −.14613653E+03 −8.1187 25.89
1.45 −.14765363E+03 −8.20298 25.89
1.50 −.14881189E+03 −8.26733 25.89
1.55 −.14968276E+03 −8.31571 25.89
1.60 −.15032905E+03 −8.35161 25.89
1.65 −.15080045E+03 −8.3778 25.89
1.70 −.15113533E+03 −8.39641 25.89
1.75 −.15136489E+03 −8.40916 25.89
1.80 −.15151354E+03 −8.41742 25.89
1.85 −.15159954E+03 −8.4222 25.89
1.90 −.15163737E+03 −8.4243 25.89
1.95 −.15163834E+03 −8.42435 25.89
2.00 −.15161094E+03 −8.42283 25.89
2.05 −.15156141E+03 −8.42008 25.89
2.10 −.15149510E+03 −8.41639 25.89
2.15 −.15141609E+03 −8.41201 25.89
2.20 −.15132742E+03 −8.40708 25.89
2.25 −.15123176E+03 −8.40176 25.89
2.30 −.15113038E+03 −8.39613 25.89
2.35 −.15102515E+03 −8.39029 25.89
2.40 −.15090103E+03 −8.38339 25.89
2.45 −.14935582E+03 −8.29755 25.89
2.50 −.14764123E+03 −8.20229 25.89
2.55 −.14627167E+03 −8.1262 25.89
2.60 −.14356170E+03 −7.97565 25.89
2.65 −.14113231E+03 −7.84068 25.89
2.70 −.13868309E+03 −7.70462 25.89
2.75 −.14196237E+03 −7.8868 25.89
2.80 −.13859814E+03 −7.6999 25.89
2.85 −.13527883E+03 −7.51549 25.89
2.90 −.13183631E+03 −7.32424 25.89
2.95 −.12797128E+03 −7.10952 25.89
3.00 −.13160835E+03 −7.31158 25.89
EOF

gnuplot
set terminal svg enhanced
set xlabel 'Lattice Parameter'
set ylabel 'Energy (eV)'
set out 'Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_EV_data.svg'; p '
    Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_EV_data.dat' u 1:3
q

qlmanage −t −s 1000 −o . Slab_TiN_111_N−terminated_TiN_Dimer_Ti_down_11x11x1_EV_data
    .svg

clear



 Writing file EV_Plotter.sh...
  done
```

```


300 +---------------------------------------------------------------------------+
    |*          +             +           +           +             +           |
    |*                                                                  ******* |
    |*                                                                          |
    |*                                                                          |
    |*                                                                          |
250 |*+                                                                       +-|
    |*                                                                          |
    |*                                                                          |
    |*                                                                          |
    |*                                                                          |
200 |-*                                                                       +-|
    | *                                                                         |
    | *                                                                         |
    | *                                                                         |
    | *                                                                         |
    | *                                                                         |
150 |-*                                                                       +-|
    | *                                                                         |
    | *                                                                         |
    | *                                                                         |
    | *                                                                         |
    |  *                                                                        |
100 |-+*                                                                      +-|
    |  *                                                                        |
    |   *                                                                       |
    |   *                                                                       |
    |   *                                                                       |
    |    *                                                                      |
 50 |-+  *                                                                    +-|
    |     *                                                                     |
    |      **                                                                   |
    |        *                                                                  |
    |         **                                                                |
  0 |-+         *******                                                       +-|
    |                  *********************************************************|
    |                                                                           |
    |                                                                           |
    |                                                                           |
    |           +             +           +           +             +           |
-50 +---------------------------------------------------------------------------+
    0          0.5           1          1.5           2            2.5          3
                                 Lattice Parameter


 VASP data collection and compilation completed
 Success, End of Script
```

9.9 A script to collect and collate many vasprun.xml files with for loops, and a temporary counter file

As we talked about in section 9.8, VASP calculations create many files of output. One such file is called vasprun.xml and contains a summary of what was used for the calculation input, forces, stresses, charges, and other properties. Collections of many vasprun.xml files are used to generate force field files with codes like MEAMfit.

MEAMfit requires a collection of vasprun.xml files all with individual names like vasprun1.xml, vasprun2.xml, ... Since VASP by default outputs a single vasprun.xml file of exactly that name into a calculation directory, we must collect, collate, and rename many of these vasprun.xml files into a single directory that MEAMfit can access and use. This could be done manually but quickly becomes tedious when the number of vasprun.xml files becomes large. This can be quickly automated with a shell script. A sample of the contents of a directory containing several VASP calculations and the contents of a single directory of those calculations is included as follows:

```
uname@uname:~O2> ls
O2_Dimer_a0_1.183   O2_Dimer_a0_1.195   O2_Dimer_a0_1.207   O2_Dimer_a0_1.219
    O2_Dimer_a0_1.231   README.txt   test_O2_dimer_automated_vasp.sh
uname@uname:~O2> cd O2_Dimer_a0_1.183/
uname@uname:~O2/O2_Dimer_a0_1.183> ls
CHG   CONTCAR   EIGENVAL   INCAR   OUTCAR   POSCAR   REPORT   XDATCAR   vasprun.xml   CHGCAR
      DOSCAR    IBZKPT     KPOINTS OSZICAR  PCDAT    POTCAR   WAVECAR   slurm-45241924.out
```

To achieve a script with this functionality, we will assume that many calculations with VASP have been done and therefore rely on general automatic file naming conventions developed in the previous examples from sections 9.5, and 9.4. The following script will collect the vasprun.xml files from a number of calculation directories, automatically rename them, and place them into a directory of your choosing.

As a warning, this is generally a super brute-force way of running this file collection! I am including it for completeness of this text but a much more streamlined script to achieve similar ends is included and discussed at length in section 10.4. For the more general case of collecting many files across many levels of directories, please refer to the script in section 10.4.

In order to simplify the input parameters of the for loop that does the main work in this calculation, we will be borrowing the header and input variables from the script given in section 9.4; as such, please refer to that section for a more in-depth discussion of how the header and initial variables created in that script are working.

This script uses the file naming convention shown in section 9.4 in a for loop that iteratively enters directories, refers to a temporary file that has a number stored inside it, copies the vasprun.xml file to a target file and simultaneously renames it based on the contents of the temporary file, navigates back to the parent directory, and then increases the number in the temporary file by 1. This use of the temporary file is beneficial in some cases as compared to strictly using a variable to contain this value because the temporary file can be referred to over many executions of this script. That is to say we can continue the naming of these files picking up where we left off for many sets of vasprun.xml files (which is exactly what frequently needs to be done when creating force field files). This could also be achieved by modifying counter variable parameters in your scripts each time you run them but I also find that to be tedious. Please see the following script:

```
System_Name="O2_Dimer"  # Give a calculation title for VASP

Lattice_Parameter="1.208"
# Give the real or experimental minimum energy interatomic distance between oxygen
    atoms
Calculation_Steps_in_Each_Direction="2"
# Give the total number of calculation points in each direction (e.g. 20 in the +
    direction, 20 in the - direction = 41 total including a0)
Lattice_Parameter_Variation="2"
# Give upper and lower bounds of % a0 change e.g., a value of 2 would mean that you
    want a 2% variation which is a0 +/- 0.02*a0
```

```bash

File_Number_Start="1"
# Give the number that you want to begin appending to the vasprun.xml files: e.g., vasprun1.xml

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:- BASH and bc Calculated Variables for VASP Automation :-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Number_of_Jobs=$(echo "scale=2;(($Calculation_Steps_in_Each_Direction*2)+1)" | bc)
Lat_Param_Max=$(echo "scale=2;($Lattice_Parameter)*(    1.0 $Lattice_Parameter_Variation)" | bc)
# Use bc to calculate a0_max
Lat_Param_Min=$(echo "scale=2;($Lattice_Parameter)*(1 - 0.0 $Lattice_Parameter_Variation)" | bc)
# Use bc to calculate a0_min
Step_Size=$(echo "scale=3;( (($Lat_Param_Max) - ($Lat_Param_Min))/( $Calculation_Steps_in_Each_Direction*2) )" | bc)
# Use bc to calculate N_steps as step size

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-: Begin Main for() Loop for VASP File Collection -:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

touch tempfile; echo ${File_Number_Start} > tempfile
# Create an empty temp file and fill it with the number starting file number
mkdir ${System_Name}_vasprun_xml
# Create an empty target directory for copied and renamed files
echo " Collecting a total of ${Number_of_Jobs} vasprun.xml files from the ${System_Name} calculations..."

for lat in `seq -w ${Lat_Param_Min} ${Step_Size} ${Lat_Param_Max}`; do
# ` is a backtick: Everything between backticks is executed by the shell before the main command, output is then used by that command
  file_number=$(cat tempfile)   # Extract the current number in the tempfile
    cd ${System_Name}_a0_${lat} # Navigate into directory for each loop iteration
    cp vasprun.xml ../${System_Name}_vasprun_xml/vasprun${file_number}.xml
    # Copy and rename the vasprun.xml file to the target directory based on the current number in the tempfile
    echo " Copied vasprun.xml to vasprun${file_number}.xml"
    cd .. # Navigate back to the parent directory
  counter=$[$(cat tempfile) + 1]
  # Increment the counter + 1 for the next step in the loop, inspiration for this tempfile implementation is thanks to StackOverflow user bos
  echo $counter > tempfile
  # Rewrite the temp file with cat and a redirect with the current value of the counter
  echo " Reset file naming counter, new value is ${file_number}"
  # Iterate until for loop conditions are exhausted
done  # End of main for() loop

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-: End Main for() Loop for VASP File Creation -:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " vasprun.xml file collection and renaming completed"
echo " Success, End of Script"
```

A sample way to run this script, as well as the output printed to the terminal from running this script, and the new contents of the directory after collecting, renaming, and collating all of the vasprun.xml files is shown in the following. In this example, the script is run on the directory shown above that contains several VASP calculation output directories. The calculations are for the ground state energy in an O_2 dimer for several different bonding lengths.

```
uname@uname:~/calc/vasp_meam/test_O2_dimer_automated_vasp> File_Name="VASP_Run_File_Collector"; touch ${File_Name}.sh; chmod +x ${File_Name}.sh; vim ${File_Name}.sh; ./${File_Name}.sh
Collecting a total of 5 vasprun.xml files from the O2_Dimer calculations...
 Copied vasprun.xml to vasprun1.xml
 Reset file naming counter, new value is 1
 Copied vasprun.xml to vasprun2.xml
 Reset file naming counter, new value is 2
 Copied vasprun.xml to vasprun3.xml
 Reset file naming counter, new value is 3
 Copied vasprun.xml to vasprun4.xml
 Reset file naming counter, new value is 4
 Copied vasprun.xml to vasprun5.xml
 Reset file naming counter, new value is 5
 vasprun.xml file collection and renaming completed
 Success, End of Script
```

```
uname@uname:~/calc/vasp_meam/test_O2_dimer_automated_vasp/O2_Dimer_vasprun_xml> ls
vasprun1.xml  vasprun2.xml  vasprun3.xml  vasprun4.xml  vasprun5.xml
```

9.10 A Gnuplot script for plotting OSZICAR and ignoring the file header

In many cases, the header line of a file contains information about what the data within is specifically so there isn't a bunch of ambiguity when someone reviews the contents of the file(s). However, without telling Gnuplot that the file has a header or doing some tricks with covering all the text with quotation marks, Gnuplot will not play nicely. This is especially apparently when you are trying to plot many files in rapid succession without modifying them by hand. In order to do this reasonably easily, you can tell Gnuplot to use the header of the file as if it were the titles of the columns and Gnuplot will happily use the text as such.

This is immediately applicable when you are trying to plot the convergence criteria in an OSCIZAR file that VASP will create. Even more so, if you are trying to rapidly visualize the convergence criteria for many of these OSZICAR files in rapid succession, then you'll probably want to automate as much as possible. Please see the following Gnuplot script:

```
# Shows iteration vs. E0 convergence from an OSZICAR file in the working directory
gnuplot
set terminal dumb size 105,55
set key autotitle columnhead; unset key # Ignores first line. Thanks to Matthew on StackOverflow
set xlabel 'Lattice Parameter'; set ylabel 'Energy (eV)'
p 'OSZICAR' u 1:3 with lines
q
```

10 Shell Scripting for Force Field Creation with MEAMfit and MEAMfit2

MEAMfit [16] and MEAMfit2 are programs that are capable of fitting an EAM or MEAM potential to a set of vasprun.xml atomic configurations. This is especially useful in the case that you cannot find a suitable interatomic potential to accurately describe the system that you are considering or if you want to develop a custom force field for your own purposes. These force fields are applicable in molecular dynamics calculations, especially with programs like Sandia National Lab's LAMMPS.

10.1 Compiling MEAMfit

MEAMfit, as compared with some other codes, is actually straightforward to compile and uses a csh script to do all of the heavy lifting for you. Please see a terminal transcript of the compilation and checking the compiled executable for MEAMfit version 1.02:

```
uname@uname:~MEAMfit> ls
MEAMfit.tar.gz
uname@uname:~MEAMfit> untar MEAMfit.tar.gz
uname@uname:~MEAMfit> ls
MEAMfit.tar.gz  MEAMfitUserGuide.pdf  README  SampleCalculation  src
uname@uname:~MEAMfit> cd src/
uname@uname:~MEAMfit/src> ls
installmeam            m_datapoints.f90        m_filenames.f90      m_geometry.f90        m_neighborlist.f90   m_plotfiles.f90   m_screening.f90   source.f90
m_atomproperties.f90   m_electrondensity.f90   m_general_info.f90   m_meamparameters.f90   m_optimization.f90   m_poscar.f90      old.f90
uname@uname:~MEAMfit/src> csh installmeam
compiler: ifort
source:  m_atomproperties.f90 m_datapoints.f90  m_electrondensity.f90 m_filenames.f90 m_general_info.f90  m_geometry.f90 m_meamparameters.f90 m_neighborlist.f90 m_optimization.f90 m_plotfiles.f90 m_poscar.f90 m_screening.f90  source.f90
compilation flags: -g -assume byterecl -fltconsistency -fpconstant -real-size 64 -O3
libraries:
ifort m_atomproperties.f90 m_datapoints.f90 m_electrondensity.f90 m_filenames.f90 m_general_info.f90 m_geometry.f90 m_meamparameters.f90 m_neighborlist.f90 m_optimization.f90 m_plotfiles.f90 m_poscar.f90 m_screening.f90 source.f90 -g -assume byterecl -fltconsistency -fpconstant -real-size 64 -O3   -o MEAMfit
source.f90(10025): remark #8291: Recommended relationship between field width 'W' and the number of fractional digits 'D' in this edit descriptor is 'W>=D+7'.
          write(*,'(A9,E10.5,A21)') ' OPTDIFF=',optdiff,' (from settings file)'
-------------------------^
chmod ugo+r *
chmod ugo+x MEAMfit
uname@uname:~MEAMfit/src> ls
MEAMfit       m_atomproperties.f90   m_datapoints.mod        m_filenames.f90      m_generalinfo.mod   m_meamparameters.f90   m_neighborlist.mod   m_plotfiles.f90   m_poscar.mod      old.f90
Makefile      m_atomproperties.mod   m_electrondensity.f90   m_filenames.mod      m_geometry.f90      m_meamparameters.mod   m_optimization.f90   m_plotfiles.mod   m_screening.f90   source.f90
installmeam   m_datapoints.f90       m_electrondensity.mod   m_general_info.f90   m_geometry.mod      m_neighborlist.f90     m_optimization.mod   m_poscar.f90
```

```
    m_screening.mod
uname@uname:~MEAMfit/src> file MEAMfit
MEAMfit: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
    interpreter /lib64/l, BuildID[sha1]=3dd8xf2gge3a1f286fdbfc36g4309a8e6735436b,
    for GNU/Linux 3.2.0, with debug_info, not stripped
uname@uname:~MEAMfit/src> ./MEAMfit

 ------------- MEAMfit (version 1.02) -------------
 By Andrew I. Duff and Marcel H. F. Sluiter, 2006-2015
 --------------------------------------------------
ls: cannot access 'vasprun*.xml': No such file or directory

 Finished writing fitdbse file, stopping.
```

10.2 Compiling MEAMfit2

Compiling MEAMfit2 is similar in complexity and method to compiling MEAMfit but perhaps slightly more fiddly. Being the more difficult case, I will include the terminal transcript of my experience here. Suffice it to say that the most up to date Intel FORTRAN compilers should do the trick:

```
uname@uname:~/codes/MEAMfit2/src> module list
Currently Loaded Modulefiles:
  1) modules/3.2.11.4 2) altd/2.0 3) darshan/3.2.1 4) craype-network-aries 5) craype
    -haswell 6) craype-hugepages2M 7) intel/19.0.3.199 8) craype/2.6.2 9) cray-mpich
    /7.7.10 10) cray-libsci/19.06.1 11) udreg/2.3.2-7.0.1.1_3.57__g8175d3d.ari 12)
    ugni/6.0.14.0-7.0.1.1_7.59__ge78e5b0.ari 13) pmi/5.0.14 14) dmapp/7.1.1-7.0.1.1
    _4.68__g38cf134.ari 15) gni-headers/5.0.12.0-7.0.1.1_6.44__g3b1768f.ari 16)
    xpmem/2.2.20-7.0.1.1_4.26__g0475745.ari 17) job/2.2.4-7.0.1.1_3.53__g36b56f4.ari
     18) dvs/2.12_2.2.167-7.0.1.1_17.9__ge473d3a2 19) alps/6.6.58-7.0.1.1_6.26
    __g437d88db.ari 20) rca/2.2.20-7.0.1.1_4.70__g8e3fb5b.ari 21) atp/2.1.3 22)
    PrgEnv-intel/6.0.5
uname@uname:~/codes/MEAMfit2/src> csh installmeam
compiler: ifort
source:  m_atomproperties.f90 m_datapoints.f90  m_electrondensity.f90 m_filenames.
    f90 m_general_info.f90  m_geometry.f90 m_meamparameters.f90 m_neighborlist.f90
    m_optimization.f90 m_plotfiles.f90 m_poscar.f90 m_screening.f90
    m_objectiveFunction.f90 m_observables.f90  source.f90
compilation flags: -g -assume byterecl -fltconsistency -fpconstant -real-size 64 -O3
libraries:
ifort m_atomproperties.f90 m_datapoints.f90 m_electrondensity.f90 m_filenames.f90
    m_general_info.f90 m_geometry.f90 m_meamparameters.f90 m_neighborlist.f90
    m_optimization.f90 m_plotfiles.f90 m_poscar.f90 m_screening.f90
    m_objectiveFunction.f90 m_observables.f90 source.f90 -g -assume byterecl -
    fltconsistency -fpconstant -real-size 64 -O3  -o MEAMfit
source.f90(30679): remark #8291: Recommended relationship between field width 'W'
    and the number of fractional digits 'D' in this edit descriptor is 'W>=D+7'.
          if (printSO) write(*,'(A9,E10.5,A21)') ' OPTDIFF=',optdiff,' (from
    settings file)'
----------------------------------------^
chmod ugo+r *
chmod ugo+x MEAMfit
uname@uname:~/codes/MEAMfit2/src> ls
LICENSE.txt   dev2dist.sh   m_atomproperties.f90   m_datapoints.mod        m_filenames.
    f90       m_generalinfo.mod   m_meamparameters.f90   m_neighborlist.mod
    m_observables.f90    m_optimization.mod   m_poscar.f90       m_screening.mod
```

```
MEAMfit        header.txt    m_atomproperties.mod   m_electrondensity.f90   m_filenames.
    mod       m_geometry.f90       m_meamparameters.mod   m_objectiveFunction.f90
    m_observables.mod     m_plotfiles.f90       m_poscar.mod       source.f90
Makefile       installmeam    m_datapoints.f90        m_electrondensity.mod
    m_general_info.f90   m_geometry.mod       m_neighborlist.f90      m_objectivefunction
    .mod   m_optimization.f90   m_plotfiles.mod       m_screening.f90
uname@uname:~/codes/MEAMfit2/src> file MEAMfit
MEAMfit: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
    interpreter /lib64/l, BuildID[sha1]=a19aeaa772949d1d5594fa8f252d633317ce3078,
    for GNU/Linux 3.2.0, with debug_info, not stripped
uname@uname:~/codes/MEAMfit2/src> ./MEAMfit
   __  __ ______     _     __  __  __ _ _   ____
  |  \/  | _____|   / \   |  \/  |/ _(_) |_|___ \
  | |\/| |  _|    / _ \  | |\/| | |_| | __| __) |
  | |  | | |___ / ___ \| |  | |  _| | |_ / __/
  |_|  |_|_____/_/   \_\_|  |_|_| |_|\__|_____|
          -------- Version 1.06 --------

               andrew.duff@stfc.ac.uk
                Copyright 2018 STFC

                     Authors:
                  Andrew I. Duff
               Marcel H. F. Sluiter

              Contributing authors:
       Prashanth Srinivasan, Thomas Mellan
          Yuelei Bai, Blazej Grabowski

      Please cite the following article if
     you use this code in your publication:

       "MEAMfit: A reference-free modified
      embedded atom method (RF-MEAM) energy
             and force-fitting code"
      A. I. Duff, et al, Comp. Phys. Comm.
                196 439-445 (2015)

 Scanning directory with ls to find vasprun.xml and .geom (castep) files
ls: cannot access 'vasprun*.xml': No such file or directory
ls: cannot access '*.geom': No such file or directory
 Cannot find input DFT files (vasprun#.xml and #.geom accepted,
 where # is any alphanumeric sequence). Stopping.
uname@uname:~/codes/MEAMfit2/src>
```

10.3 Some common error sources in MEAMfit

As I've discusses before, I encounter all sorts of errors when I run calculations, especially when I am trying a new-to-me code. Here is a partial list of some error sources and potential solutions that I have found for use with MEAMfit and MEAMfit2. I am including these items with the presentation first of the error you may see, and then a way you may attempt to resolve the error. Some of these errors may be able to resolved in other ways, however this list comprises my best knowledge on how to resolve the issues.

Error	Potential Resolution
MEAMfit or MEAMfit2 halts unexpectedly without an error report or any indication that something unexpected has happened	MEAMfit or MEAMfit2 may have run out of physical memory. This can be a serious issue if you are trying to run multiple instances on a single chip with limited memory. Try to increase the memory allocated to each instance of MEAMfit or MEAMfit2
If the R_{fit} value does not converge to a low number (e.g. perhaps it stays near $R_{fit} = 5$ or something similarly large even after many hours of fitting)	Check and see if your vasprun.xml files have large energies, MEAMfit and MEAMfit2 do not seem to like energies that are on the order of anything greater than $10^3 eV$ when your equilibrium energies are closer to the $1eV$ or $0.1eV$ level. If you have energies on the order of $10^5 eV$ or so, you may find very sluggish convergence
When trying to run a continued calculation, MEAMfit or MEAMfit2 will halt with an error saying that some of the best optimization function files are not readable or not existing	MEAMfit or MEAMfit2 may have halted unexpectedly, it seems that continued calculations are only supported if the program halts on its own accord (e.g., by timing out based on the maximum calculation time given in the settings file). Try to run MEAMfit or MEAMfit2 for several hours less than the maximum allotment that you can get so that the program will not halt unexpectedly
MEAMfit2 will give huge numbers of errors like 'negative electron density' errors and 'Optimization failed: F cannot be evaluated (F=NaN)' printed to the terminal (or the .out file)	The best workaround for this error, I have found, is to switch back to the older version MEAMfit 1.02 which does not seem to have the same issue. There may be other fixes or this could be related to a compilation issue, however the original version seems to be robust to this particular problem and I therefore prefer it

10.4 Collecting many vasprun.xml files for use with MEAMfit by using for loops and the expr command

MEAMfit requires input of vasprun.xml files in the standard use case. Collecting many of these files, say several thousand, and renaming them all to reflect something usable by MEAMfit, is impractical to do by hand. Being such, this is an excellent use case of automation with the shell! What follows is a much more streamlined and more broadly applicable version of the script given in section 9.9

The script what we'll discuss in this section is nice because it will navigate into a series of directories and extract all of the vasprun.xml files by copying them to a new location with a new unique name that MEAMfit will be able to recognize. Note that this script, while flexible enough to easily be re-written to suit individual needs, is set up with the following directory tree structure in mind (please see the following plain text diagram of the directory tree that this script works for):

```
                    Top Directory:
          +-----------+       +------------+
          | Crystals  |       |  Molecules |
          +-----------+       +------------+

              Second Level Directories:
        +----+ +------+ +-----+     +---+ +---+
        |TiN| |Al2O3| |TiO2|     |N2| |O2|
        +----+ +------+ +-----+     +---+ +---+

              Third Level Directories:
        +------------------------------------+
        |Individual Calculations 1...2...n|
        +------------------------------------+
```

For more coverage of this sort of all-text diagram, please see section 12.2.

The script expects that broad categories of calculations are all grouped into their own directories e.g., Crystals, Molecules, Slabs, etc... Below that in the directory tree, the script expects that all of the individual systems have their own directories e.g., one for TiN in the Crystals directory, one for O2 in the Molecules directory, etc... And then finally, at the third level of the directory tree, the script expects that all of the individual VASP calculations (which will contain their own vasprun.xml file) are contained within their own directories inside of their parent directory.

Overall, the script uses a triplet of nested for loops to do the heavy lifting. In all of the for loops, there is a single variable: a, b, and c defined: one for each loop. Then it says: for variable (e.g., a, b, c) in all of the directories within the current directory, do... Then, within the center of all the for loops, the program will copy the vasprun.xml file up three directories to the top of the file tree within a new directory that is made to contain all of the vasprun.xml files. The final piece of the puzzle is to rename the vasprun files so that they are not redundant and are also recognizable by MEAMfit. This is done simply by creating a variable called File_Number at the beginning of the script which we set to 1, and then for every iteration of the for loops, we add 1 to the number. That functionality of adding one is done using the expr command in BASH. This is acceptable (since expr doesn't like to handle floating point arithmetic) because we only need to do this operation for integers.

Please see the following script

```
mkdir vasprun_files
File_Number="1" # Begin a variable to use in the for() loop to count the current job
     up to the final job

for a in */ ; do # Loop starting at top directory. Contents: Crystals, Molecules,
    etc...
  echo "Entering directory $a"
  cd "$a" ; pwd

    for b in */ ; do # Loop within a major calculation category directory
      echo "Entering directory $b"
      cd "$b" ; pwd

          for c in */ ; do # Loop within a single calculation directory
            echo "Entering directory $c"
            cd "$c" ; pwd

              cp vasprun.xml ../../../vasprun_files/vasprun${File_Number}.xml # Copy
     and rename the vasprun.xml file to the target directory based on the current
    number in the tempfile
                echo " Copied vasprun.xml to vasprun${File_Number}.xml"
                File_Number=$(echo `expr $File_Number + 1`)  # Add 1 to the
    Current_Job_Number counter variable

            cd ..
          done # End of single calculation directory for() loop

      cd ..
    done # End of major calculation category directory for() loop

  cd ..
done  # End of main for() loop   # Iterate until for loop conditions are exhausted

echo " vasprun.xml file collection and renaming completed"
echo " Success, End of Script"

```

10.5 Plotting a MEAMfit interatomic separations histogram in Gnuplot

For the purposes of this guide, I will be referring to MEAMfit2 because of its advantages in terms of speed in some arenas compared to the original versions of MEAMfit.

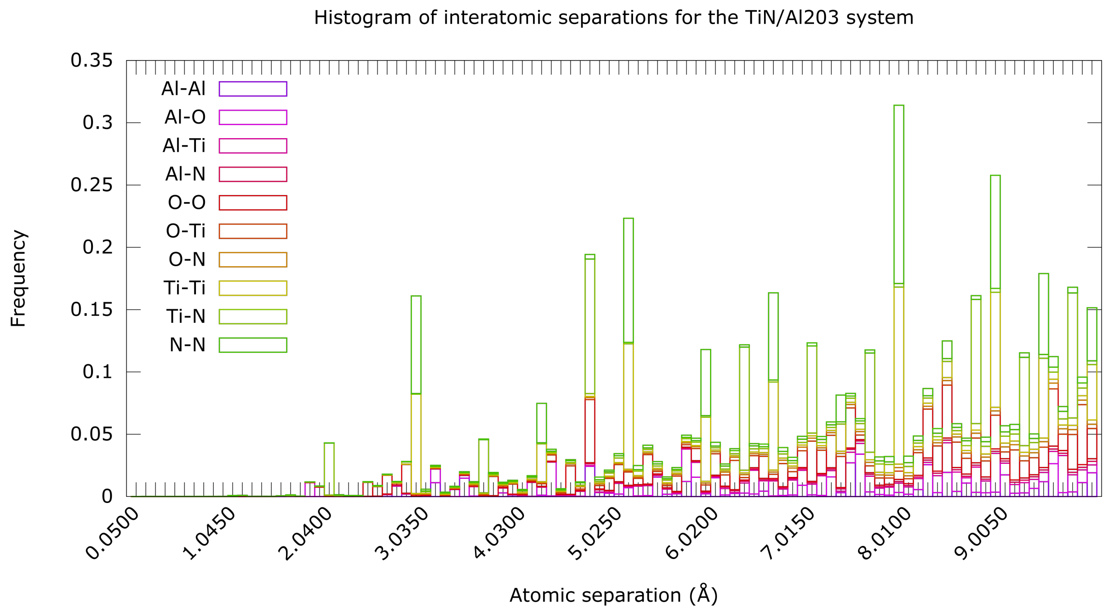
Running MEAMfit2 for the first time in a directory containing a number of vasprun.xml files, MEAMfit2 will generate a file called fitdbse. In the case that there are more than 1000 vasprun.xml files, you will have to edit the fitdbse file's first few characters to include the number of vasprun files that you want MEAMfit2 to consider. Unlike the original MEAMfit program however, MEAMfit2 will automatically populate this with the number of files that you have in the case that there are less than 1000.

In the case that you are trying to model a system that has directional bonding (like in covalent materials), you will want to create a MEAM file instead of an EAM file. This option can be selected when running MEAMfit2 for the second time which will create a file called settings. You will need to edit the settings file to 1) uncomment (by removing the # sign) the CUTOFF_MAX= line and add a suitable CUTOFF_MAX value. You can determine a suitable CUTOFF_MAX value from the histogram of interatomic separations that MEAMfit2 will generate for you if you run the program with the noopt (no-optimization) option. You will also need to add the string STOPTIME=24 (or some other relatively large amount of time where the number following STOPTIME is hours). These edits can be made easily using vim as described in section 3.7.

If you are finding that your program is quitting earlier than you might expect, examine the logs. If you find that there is an issue saying something like '...there is no interatomic separation information for species 2 and 4' then you might have neglected to include vasprun.xml files for the interactions of some of the species in your range of elements that you are including. For example, if you are trying to make a MEAM file for Al, O, Ti, and N, you need interaction information for all of the possible first order interactions.

For the reader's reference, I am including here some notes on this histogram of interatomic separations that MEAMfit2 will generate. The file will be of plain text data in columns, the first being a distance unit and all remaining columns will be the interatomic interactions. These interactions are split into series based on atom pair type. For example, if you had 2 species (A, and B) then there would be three columns following the first, being the interaction of A with A, then the interaction of A with B, and finally the interaction of B with B. For the case of this tutorial, I ran calculations for Al, O, Ti, and N so my histogram of interatomic separations had 11 columns.

I have included below a plot of interatomic separations in the systems that I calculated for Al, O, Ti, and N. The specific frequency of occurrence is slightly misleading because there are in some cases more interactions of some species with others just based on the configurations that I calculated. All that you need to consider is that each of the bins represents an order of nearest neighbor interactions in your system. What you need to decide is where there is a good cutoff point of diminishing returns for distance that you are calculating vs. accuracy you care about.



Following is a Gnuplot script (see section 3.8 for more on Gnuplot) for plotting the nearest-neighbor histogram output of MEAMfit as is shown above. This is a slightly more advanced usage of Gnuplot than I have shown in some places in this text because it fiddles with the column commands that are actually being passed to Gnuplot. In this example, if it were not for the u 2:xtic(int($0)%10 == 0 ? stringcolumn(1) : ”) commands, then the x-axis would be over-populated with labels of the data set. The 10% command in the following Gnuplot script tells the program that you only want 1 in 10 of the labels to be printed to the final graphic. Lines 3, and 4 tell Gnuplot that the data is already in a histogram format and that we want to arrange the histogram bins as boxes that are all stacked on top of each other. All of the strings saying lw 2 lc rgb '#8F00E5' tell a width of the line that we want and then give the lines a specific hex color. For a table of hex colors that can be used for data sets like this, please see section 3.8. This means of plotting can be applied to all manner of data that is arranged as a histogram.

```
1 gnuplot
2 set terminal svg enhanced size 1929,1080 font 'Verdana,32'
3 set style data histograms
4 set style histogram rowstacked
5 set xtics rotate by 45 right
6 set key left; set key font ",32"
```

```
set xlabel 'Atomic separation (A)'; set ylabel 'Frequency'; set title 'Histogram of
    interatomic separations for the TiN/Al203 system'
set out 'sepnHistogram.svg'
# Thanks to Christoph from StackOverflow for the xtic(...) example c. 2016
p 'sepnHistogram.out' \
   u 2:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'Al-Al' lw 2 lc rgb '#8F00E5'
    ,\
'' u 3:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'Al-O'  lw 2 lc rgb '#D800E0'
    ,\
'' u 4:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'Al-Ti' lw 2 lc rgb '#DC009B'
    ,\
'' u 5:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'Al-N'  lw 2 lc rgb '#D8004F'
    ,\
'' u 6:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'O-O'   lw 2 lc rgb '#D40007'
    ,\
'' u 7:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'O-Ti'  lw 2 lc rgb '#CF3D00'
    ,\
'' u 8:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'O-N'   lw 2 lc rgb '#CB8000'
    ,\
'' u 9:xtic(int($0)%10 == 0 ? stringcolumn(1) : '')  t 'Ti-Ti' lw 2 lc rgb '#C7C000'
    ,\
'' u 10:xtic(int($0)%10 == 0 ? stringcolumn(1) : '') t 'Ti-N'  lw 2 lc rgb '#88C300'
    ,\
'' u 11:xtic(int($0)%10 == 0 ? stringcolumn(1) : '') t 'N-N'   lw 2 lc rgb '#46BF00'
    ; q

```

The plain text data file for recreating the above plot is included below with columns of interatomic separation followed by all of the possible first order interactions e.g., Al-Al, Al-O, Al-Ti, Al-N, O-O, O-Ti, O-N, Ti-Ti, Ti-N, N-N:

```
# sepn   no. sepns within range, per species (e.g. (1,1); (1,2); (2,2))
0.0500  0.00000  0.00000  0.00000  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.1495  0.00001  0.00001  0.00001  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.2490  0.00001  0.00001  0.00001  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.3485  0.00001  0.00001  0.00001  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.4480  0.00001  0.00001  0.00001  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.5475  0.00001  0.00001  0.00001  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.6470  0.00001  0.00001  0.00001  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.7465  0.00001  0.00001  0.00001  0.00001  0.00000  0.00001  0.00001  0.00001  0.00001
    0.00002
0.8460  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001
    0.00002
0.9455  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00002
    0.00002
1.0450  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00001  0.00002
    0.00076
1.1445  0.00001  0.00001  0.00001  0.00001  0.00070  0.00001  0.00011  0.00001  0.00002
    0.00005
1.2440  0.00001  0.00001  0.00001  0.00001  0.00002  0.00002  0.00002  0.00001  0.00002
    0.00003
```

```
1.3435 0.00001 0.00001 0.00001 0.00002 0.00001 0.00002 0.00003 0.00002 0.00003
    0.00004
1.4430 0.00001 0.00001 0.00001 0.00001 0.00001 0.00002 0.00002 0.00002 0.00002
    0.00003
1.5425 0.00001 0.00041 0.00001 0.00002 0.00001 0.00002 0.00002 0.00002 0.00003
    0.00004
1.6420 0.00004 0.00030 0.00004 0.00005 0.00001 0.00003 0.00004 0.00003 0.00071
    0.00006
1.7415 0.00003 0.00005 0.00002 0.00004 0.00001 0.00003 0.00004 0.00003 0.00004
    0.00005
1.8410 0.00003 0.01107 0.00002 0.00021 0.00005 0.00015 0.00005 0.00003 0.00004
    0.00006
1.9405 0.00003 0.00750 0.00002 0.00004 0.00006 0.00059 0.00006 0.00003 0.00007
    0.00006
2.0400 0.00003 0.00004 0.00005 0.00005 0.00006 0.00080 0.00008 0.00008 0.04175
    0.00010
2.1395 0.00003 0.00003 0.00007 0.00008 0.00008 0.00034 0.00013 0.00013 0.00034
    0.00020
2.2390 0.00002 0.00003 0.00006 0.00006 0.00009 0.00010 0.00012 0.00009 0.00013
    0.00016
2.3385 0.00003 0.00003 0.00005 0.00006 0.00008 0.00009 0.00011 0.00009 0.00014
    0.00017
2.4380 0.00003 0.00004 0.00005 0.00006 0.01114 0.00012 0.00012 0.00008 0.00013
    0.00017
2.5375 0.00004 0.00011 0.00006 0.00008 0.00768 0.00014 0.00015 0.00011 0.00024
    0.00022
2.6370 0.00148 0.00004 0.00039 0.00007 0.01505 0.00014 0.00015 0.00027 0.00017
    0.00020
2.7365 0.00834 0.00005 0.00088 0.00008 0.00130 0.00016 0.00017 0.00070 0.00018
    0.00021
2.8360 0.00195 0.00007 0.00057 0.00007 0.02306 0.00016 0.00017 0.00172 0.00016
    0.00020
2.9355 0.00018 0.00007 0.00016 0.00018 0.00111 0.00036 0.00039 0.07972 0.00044
    0.07836
3.0350 0.00041 0.00016 0.00025 0.00024 0.00062 0.00039 0.00043 0.00140 0.00052
    0.00145
3.1345 0.01111 0.01098 0.00020 0.00018 0.00023 0.00029 0.00035 0.00063 0.00037
    0.00072
3.2340 0.00025 0.00031 0.00015 0.00019 0.00031 0.00032 0.00040 0.00064 0.00039
    0.00056
3.3335 0.00007 0.00560 0.00015 0.00017 0.00046 0.00033 0.00037 0.00027 0.00035
    0.00048
3.4330 0.01473 0.00215 0.00017 0.00019 0.00033 0.00066 0.00042 0.00032 0.00040
    0.00055
3.5325 0.00036 0.00753 0.00018 0.00034 0.00028 0.00106 0.00041 0.00063 0.00058
    0.00052
3.6320 0.00009 0.00022 0.00018 0.00048 0.00038 0.00076 0.00041 0.00043 0.04265
    0.00057
3.7315 0.00009 0.00017 0.00019 0.00024 0.01539 0.00059 0.00045 0.00049 0.00090
    0.00082
3.8310 0.00291 0.00392 0.00020 0.00023 0.00112 0.00071 0.00044 0.00059 0.00057
    0.00079
3.9305 0.00122 0.00024 0.00019 0.00025 0.00810 0.00060 0.00044 0.00084 0.00046
    0.00069
4.0300 0.00069 0.00029 0.00026 0.00029 0.00059 0.00059 0.00047 0.00126 0.00053
    0.00070
4.1295 0.00020 0.01114 0.00029 0.00026 0.00066 0.00079 0.00050 0.00144 0.00046
    0.00097
```

```
4.2290 0.00104 0.00646 0.00030 0.00038 0.00211 0.00094 0.00062 0.03047 0.00082
    0.03164
4.3285 0.00046 0.02720 0.00024 0.00048 0.00499 0.00085 0.00060 0.00068 0.00087
    0.00158
4.4280 0.00037 0.00071 0.00025 0.00044 0.00106 0.00087 0.00061 0.00056 0.00055
    0.00084
4.5275 0.00015 0.00103 0.00024 0.00033 0.02285 0.00153 0.00061 0.00052 0.00154
    0.00077
4.6270 0.00022 0.00399 0.00027 0.00038 0.00112 0.00181 0.00060 0.00070 0.00192
    0.00073
4.7265 0.02421 0.00064 0.00136 0.00090 0.05067 0.00162 0.00092 0.00225 0.10803
    0.00372
4.8260 0.00318 0.00044 0.00170 0.00072 0.00278 0.00120 0.00088 0.00121 0.00138
    0.00214
4.9255 0.00264 0.00058 0.00115 0.00054 0.00962 0.00098 0.00085 0.00242 0.00117
    0.00126
5.0250 0.00199 0.00857 0.00076 0.00055 0.01377 0.00076 0.00085 0.00371 0.00174
    0.00171
5.1245 0.00705 0.00062 0.00067 0.00066 0.01070 0.00105 0.00108 0.10072 0.00116
    0.09963
5.2240 0.00053 0.00778 0.00059 0.00063 0.00508 0.00105 0.00112 0.00357 0.00116
    0.00338
5.3235 0.00071 0.00802 0.00059 0.00065 0.02367 0.00112 0.00116 0.00194 0.00115
    0.00214
5.4230 0.01558 0.00130 0.00059 0.00063 0.00204 0.00124 0.00119 0.00195 0.00167
    0.00185
5.5225 0.00179 0.00417 0.00047 0.00059 0.00176 0.00124 0.00113 0.00205 0.00109
    0.00145
5.6220 0.00212 0.00064 0.00062 0.00072 0.01227 0.00136 0.00105 0.00181 0.00112
    0.00156
5.7215 0.01209 0.02601 0.00067 0.00077 0.00208 0.00147 0.00108 0.00223 0.00107
    0.00176
5.8210 0.01547 0.01197 0.00061 0.00071 0.00975 0.00142 0.00107 0.00259 0.00120
    0.00199
5.9205 0.00212 0.00090 0.00054 0.00077 0.00496 0.00151 0.00138 0.05144 0.00128
    0.05311
6.0200 0.01505 0.00074 0.00062 0.00070 0.01719 0.00164 0.00134 0.00186 0.00139
    0.00299
6.1195 0.00130 0.00507 0.00113 0.00080 0.00905 0.00220 0.00122 0.00127 0.00242
    0.00167
6.2190 0.00247 0.00071 0.00147 0.00082 0.02352 0.00252 0.00126 0.00154 0.00255
    0.00157
6.3185 0.00276 0.00807 0.00141 0.00093 0.00204 0.00233 0.00140 0.00214 0.09880
    0.00190
6.4180 0.00206 0.01903 0.00087 0.00111 0.00968 0.00161 0.00138 0.00251 0.00241
    0.00178
6.5175 0.00427 0.00836 0.00078 0.00115 0.01769 0.00159 0.00143 0.00315 0.00153
    0.00211
6.6170 0.00054 0.01196 0.00084 0.00110 0.00343 0.00177 0.00161 0.07067 0.00160
    0.06998
6.7165 0.00087 0.00511 0.00084 0.00104 0.01859 0.00264 0.00173 0.00337 0.00201
    0.00311
6.8160 0.00124 0.01199 0.00103 0.00100 0.00467 0.00291 0.00165 0.00236 0.00193
    0.00268
6.9155 0.00898 0.01205 0.00114 0.00109 0.01342 0.00293 0.00170 0.00211 0.00256
    0.00231
7.0150 0.00135 0.00856 0.00113 0.00110 0.03233 0.00239 0.00183 0.00223 0.06993
    0.00249
```

```
7.1145 0.00086 0.01578 0.00085 0.00102 0.01864 0.00195 0.00173 0.00249 0.00250
     0.00242
7.2140 0.01575 0.00503 0.00125 0.00108 0.02601 0.00210 0.00181 0.00244 0.00210
     0.00230
7.3135 0.00310 0.00115 0.00191 0.00104 0.02496 0.00241 0.00176 0.02182 0.00177
     0.02146
7.4130 0.02704 0.00849 0.00232 0.00109 0.03219 0.00274 0.00172 0.00328 0.00180
     0.00210
7.5125 0.03397 0.00866 0.00195 0.00105 0.00373 0.00278 0.00178 0.00456 0.00223
     0.00184
7.6120 0.00370 0.01258 0.00155 0.00125 0.00553 0.00283 0.00204 0.00603 0.07966
     0.00250
7.7115 0.00164 0.00513 0.00103 0.00128 0.00580 0.00244 0.00209 0.00668 0.00251
     0.00319
7.8110 0.00115 0.00637 0.00095 0.00132 0.00505 0.00223 0.00197 0.00672 0.00205
     0.00431
7.9105 0.00327 0.00750 0.00126 0.00166 0.00669 0.00288 0.00273 0.14210 0.00283
     0.14312
8.0100 0.00182 0.00591 0.00118 0.00148 0.00385 0.00239 0.00253 0.00497 0.00307
     0.00520
8.1095 0.00556 0.00888 0.00118 0.00137 0.01819 0.00232 0.00246 0.00299 0.00206
     0.00357
8.2090 0.02549 0.00194 0.00181 0.00170 0.03921 0.00225 0.00248 0.00409 0.00190
     0.00580
8.3085 0.00313 0.01358 0.00198 0.00166 0.02026 0.00251 0.00234 0.00261 0.00271
     0.00376
8.4080 0.01936 0.02378 0.00206 0.00173 0.04248 0.00367 0.00237 0.01299 0.00227
     0.01419
8.5075 0.00411 0.01549 0.00188 0.00182 0.02033 0.00422 0.00228 0.00274 0.00265
     0.00295
8.6070 0.01082 0.00420 0.00121 0.00182 0.01260 0.00413 0.00220 0.00384 0.00392
     0.00216
8.7065 0.00334 0.02028 0.00151 0.00193 0.01989 0.00382 0.00253 0.00519 0.09971
     0.00304
8.8060 0.00299 0.00950 0.00140 0.00180 0.01289 0.00307 0.00248 0.00614 0.00371
     0.00369
8.9055 0.00347 0.03118 0.00161 0.00195 0.02705 0.00348 0.00287 0.09232 0.00308
     0.09078
9.0050 0.01737 0.00940 0.00172 0.00182 0.00548 0.00405 0.00306 0.00551 0.00349
     0.00491
9.1045 0.00264 0.00699 0.00182 0.00184 0.02414 0.00419 0.00291 0.00562 0.00355
     0.00420
9.2040 0.00292 0.00821 0.00203 0.00173 0.01376 0.00393 0.00286 0.00537 0.07093
     0.00361
9.3035 0.00652 0.00726 0.00206 0.00161 0.01366 0.00404 0.00272 0.00539 0.00333
     0.00368
9.4030 0.01192 0.00740 0.00199 0.00196 0.01635 0.00438 0.00294 0.06402 0.00310
     0.06488
9.5025 0.02635 0.00987 0.00190 0.00233 0.04594 0.00462 0.00308 0.00572 0.00401
     0.00846
9.6020 0.00312 0.02981 0.00179 0.00221 0.01311 0.00434 0.00304 0.00490 0.00603
     0.00386
9.7015 0.00371 0.01371 0.00202 0.00239 0.02795 0.00393 0.00342 0.00593 0.10036
     0.00458
9.8010 0.01119 0.01017 0.00206 0.00220 0.04813 0.00358 0.00356 0.00619 0.00408
     0.00467
9.9005 0.01902 0.00637 0.00283 0.00212 0.02424 0.00342 0.00346 0.04441 0.00302
     0.04260
```

10.6 A MEAMfit input script for some automation, and a demonstration of MEAMfit fitting and testing on an example data set

One area where I find myself making mistakes with MEAMfit (since most of the calculation is automated after you give MEAMfit all of the vasprun.xml files that it can eat) is in handling the settings file. This input script is similar to many that I include in sections 6, 7, and 9 so you can refer to those if you want a denser point-by point coverage of all the things that this script is doing. However, I will cover just a few here:

First in this script, we set up some variables, one of which being 'Continue' that we give either a value of T or F, this is dumped into the settings file which this script will create. I find that it is exceeding helpful to create the settings file using a script like this because you can dodge having to remember the names of specific options that you need to pass to MEAMfit.

After giving those several variables, the script will create a 'settings' file for MEAMfit (which means that you can skip running MEAMfit one of the three times that you generally need to in order to have a computation begin. The settings file also will come pre-built with the time interval over which you want the calculation to run and whether or not you are asking for a continued or a from-scratch calculation. I think that this is valuable because it saves me time; make the tool once and make it well and you won't have to worry about it nearly as much in the future.

After that, under the comment "BASH and bc Calculated Variables for VASP Automation", I give some switch/case statements for choosing which executable I want (e.g., MEAMfit or MEAMfit2 which each have their costs and benefits), which processor architecture to run the calculations on, and to what queue I will submit the calculation. Since MEAMfit isn't traditionally run with multi-threading, I choose to run it in serial on a single processor on a single node which incidentally saves lots in the way of machine hours compared to running over multiple processors on a node.

After creating a README file and printing parameters to the terminal for future reference, I set up an if statement to choose how the SBATCH file will be created based on which queue in a cluster I want to submit the calculation to. I find this methodology valuable when it ends up being tedious to control all of those parameters with just variables. This is a valid means of scripting where you just list things more explicitly. After that, finally, the job is submitted to the computer. Remember however to run MEAMfit on the data set initially once to generate a fit database file.

Just like in section 7.3, I built in the small functionality where I copy and paste the following line into the terminal:

```
File_Name="MEAMfit_Optimization"; touch ${File_Name}.sh; chmod +x ${File_Name}.sh;
    vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt
```

Please see the following script:

```
#!/bin/bash

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-: MEAMfit.sh -:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

# Copy and paste the below line into the cluster terminal to make and run the script
    (paste into vim and save with :wq)
# |& tee -a README.txt auto-copies terminal outputs into the README.txt file (thanks
    to Byte Commander on Stack Exchange)
```

```

# File_Name="MEAMfit_Optimization"; touch ${File_Name}.sh; chmod +x ${File_Name}.sh;
    vim ${File_Name}.sh; ./${File_Name}.sh |& tee -a README.txt

#         __    __  ______  ______  __    __  ______ __  ______
#        /\ "-./  \/\  ___\/\  __ \/\ "-./  \/\  ___/\ \/\__  _\
#        \ \ \-./\ \ \  __\\ \  __ \ \ \-./\ \ \  __\ \ \/_/\ \/
#         \ \_\ \ \_\ \_____\ \_\ \_\ \_\ \ \_\ \ \_\ \_\  \ \_\ \ \_\
#          \/_/  \/_/\/_____/\/_/\/_/\/_/  \/_/\/_/   \/_/  \/_/

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Give the Following Variables -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

Job_Name="MEAMfit" # Give the name you want to apply to all files here

Description="MEAMfit potential fitting" # Please give a short description of the
    calculation for the README.txt file
Author="Steven E. Bopp, Materials Science & Engineering"

System_Name="MEAMfit"  # Give a calculation title for MEAMfit

Job_Time="47:10:00"       # Run time in hh:mm:ss (flex queue, > 02:00:00)
Job_Time_Min="02:00:00"   # Minimum possible run time of the script for flex queue
Job_Nodes="1"             # Number of nodes to use for the calculation
Job_Queue="shared"        # Give the queue (e.g., 'debug', 'regular', 'flex', or '
    low') low and flex queues have 50 and 75% discounts respectively for KNL
Architecture="haswell"    # SLURM setting(s): (e.g., 'haswell' or 'knl' )
Module_Name="MEAMfit"     # Give the name of the module that you want to load
Continue="T"              # (e.g. 'T' for continued, 'F' for start from scratch)
SQS_Update_Time="15"      # Give the number of seconds for update cycle to wait
Calculation_Type="Test"   # "Full" for MEAMfit or "Test" to test a MEAM file
Cutoff_Max="4.4"          # Maximum cutoff distance for the MEAMfit calculation

echo " Running MEAMfit calculations on vasprun.xml files for ${System_Name}"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-: Create MEAMfit Settings File -:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Creating settings file for ${Module_Name}"

rm settings

if [[ $Calculation_Type = Full ]]
 then

echo " Creating settings file for a full $Module_Name calculation"

cat > settings << EOF
TYPE=MEAM
CUTOFF_MAX=$Cutoff_Max
NTERMS=2
NTERMS_EMB=3
STOPTIME=45
USEREF=F
CONT=${Continue}
EOF
```

```

else

echo " Creating settings file for a $Module_Name potential testing calculation"

cp potparas_best1 potparas_MEAM

cat > settings << EOF
TYPE=MEAM
POTFILEIN=potparas_MEAM
CUTOFF_MAX=$Cutoff_Max
NTERMS=2
NTERMS_EMB=3
NOOPT=true
EOF

fi

echo " Writing input file settings..."
echo "  done"


#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:- BASH and bc Calculated Variables for VASP Automation :-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

case $Module_Name in   # This switch/case will automatically insert options into the sbatch file
  MEAMfit)   Module_Location="~/codes/MEAMfit/src/MEAMfit"  ;; # Executable location for HSW
  MEAMfit2)  Module_Location="~/codes/MEAMfit2/src/MEAMfit" ;; # Executable location for HSW
esac

case $Architecture in  # This switch/case will automatically switch the environments and executable locations for various architectures like haswell and knl
  haswell) echo " Haswell architecture selected"
           Processes_Per_Node="32" ;Threads_Per_Process="2" ;;
        # Recommended Haswell settings for 1 node and 32 MPI processes per node with 2 threads each
  knl)     echo " Knight's Landing architecture selected"
           Processes_Per_Node="64" ;Threads_Per_Process="4" ;;
        # Recommended MPI setting for 1 node. NPAR should be sqrt(Processes_Per_Node)!! NPAR = 8 gives substantial savings of ~67% compared to NPAR = 1
esac

case $Job_Queue in     # This switch/case will automatically insert options into the sbatch file
  flex)    Time_Min="#SBATCH --time-min=${Job_Time_Min}" ;; # Inserts #SBATCH --time-min=0:30:00 into sbatch
  shared)  Shared="#SBATCH --shared"                     ;; # Inserts #SBATCH --shared into sbatch
esac

Date=$(date '+%d/%m/%Y %H:%M:%S')  # Give date in day/month/year hr/min/sec thanks user1293137 from https://unix.stackexchange.com/

```

```
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-: Create file README -:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

cat > README.txt << EOF
Job Name: ${Job_Name}.sh
This is a ${Module_Name} calculation of the ${System_Name} system to calculate ${
    Description}.
Calculating for ${Job_Time} with ${Job_Nodes} job nodes on the ${Job_Queue} queue.
Using ${Architecture} node(s) with ${Processes_Per_Node} processes per node and ${
    Threads_Per_Process} threads per node for a maximum of ${Job_Time} hh:mm:ss.
Calculated by ${Author} on $Date.

A transcript of the calculation as seen from the terminal follows:

EOF

echo " Writing file README.txt..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:- Print Parameters to the Terminal :-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Running a ${Module_Name} calculation of the ${System_Name} system to
    calculate ${Description}"
echo " Calculating  for ${Job_Time} with ${Job_Nodes} job nodes on the ${Job_Queue}
    queue."
echo " Using ${Architecture} node(s) with ${Processes_Per_Node} processes per node
    and ${Threads_Per_Process} threads per node for a maximum of ${Job_Time} hh:mm:
    ss"
echo " Calculated by ${Author} on $Date."
echo " Running script ${Job_Name}.sh..."
echo " The time is currently $Date "

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:- Create SBATCH Script :-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

if [[ $Job_Queue = shared ]]
 then

echo " Creating sbatch for the ${Job_Queue} job queue"

cat > ${Job_Name}.sb << EOF
#!/bin/bash
#SBATCH --job-name=${Job_Name}
#SBATCH -q ${Job_Queue}
#SBATCH -C ${Architecture}
#SBATCH -t ${Job_Time}
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=16GB

srun --cpu_bind=cores ${Module_Location}

EOF
```

```

else

echo " Creating sbatch for the ${Job_Queue} job queue"

cat > ${Job_Name}.sb << EOF
#!/bin/bash
#SBATCH --job-name=${Job_Name}
#SBATCH -N ${Job_Nodes}
#SBATCH -C ${Architecture}
#SBATCH -q ${Job_Queue}
#SBATCH -t ${Job_Time}
${Time_Min}

#OpenMP settings:
export OMP_NUM_THREADS=1
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

module load ${Module_Name}
srun -n ${Processes_Per_Node} -c ${Threads_Per_Process} --cpu_bind=cores ${
    Module_Location}

EOF

fi

echo " Writing input file ${Job_Name}.sb..."
echo "  done"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-: Run MEAMfit Calculation with SBATCH :-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

echo " Submitting ${Job_Name}.sb via sbatch..."
echo " Running job ${Job_Name} on the ${Job_Queue} queue with ${Job_Nodes} node(s)
    per job for ${Job_Time} each"
sbatch ${Job_Name}.sb
echo "  Begin:"

#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:End Elapsed Time Measurement-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

sqs # cat out the job that has been submitted
echo " The time is currently $Date "
echo " Dumping terminal session into README.txt"
echo " Success, End of Script, Running sqs on a ${SQS_Update_Time} Second Loop"
while [ 1 ] ; do Jobs=$(sqs | wc -l); echo "`expr ${Jobs} - 1` jobs in the queue";
    sqs; date; tail slurm*; sleep ${SQS_Update_Time}; done # Queue and Slurm update
    loop

```

Determining the amount of time that MEAMfit will need to run to reach a good value of R_{fit} is complicated and more or less unpredictable because of the random sampling that the program uses to find an initial R_{fit}. However, after a suitable run time (which you will most likely have to optimize based on your own experimentation with the code), you may end up with a file of best optimization functions that looks similar to the following (a fitting of 1575 vasprun.xml files):

```
uname@uname:~/vasprun_files> head fitdbse
1575 # Files | Configs to fit | Quantity to fit | Weights
vasprun1.xml                    1-1    Fr      1
vasprun10.xml                   1-1    Fr      1
vasprun100.xml                  1-1    Fr      1
vasprun1000.xml                 1-1    Fr      1
vasprun1001.xml                 1-1    Fr      1
vasprun1002.xml                 1-1    Fr      1
vasprun1003.xml                 1-1    Fr      1
vasprun1004.xml                 1-1    Fr      1
vasprun1005.xml                 1-1    Fr      1
```

```
uname@uname:~/vasprun_files> cat bestoptfuncs
 Top           10   optfuncs:
               1  :   7.659225012101019E-002   time:          35   hours
               2  :   0.100229833269679        time:          18   hours
               3  :   0.100435201178587        time:          30   hours
               4  :   0.105494405779543        time:          25   hours
               5  :   0.147677567956181        time:           5   hours
               6  :   0.159451199802870        time:          13   hours
               7  :   0.202641083389697        time:          10   hours
               8  :   0.226031273316243        time:          19   hours
               9  :   0.252660771132752        time:           1   hours
              10  :   0.286501447533809        time:           0   hours

 Total time taken    35h 22m  0s.
```

```
uname@uname:~/vasprun_files> cat settings
TYPE=MEAM
CUTOFF_MAX=4.4
NTERMS=2
NTERMS_EMB=3
STOPTIME=44
USEREF=F
CONT=F
```

```
uname@uname:~/vasprun_files> ls
bestoptfuncs     elecdens3            pairpot13_full   potparas_best1
datapnts_best1   elecdens3_full       pairpot14        potparas_best10
datapnts_best10  elecdens4            pairpot14_full   potparas_best2
datapnts_best2   elecdens4_full       pairpot22        potparas_best3
datapnts_best3   embfunc1             pairpot22_full   potparas_best4
datapnts_best4   embfunc2             pairpot23        potparas_best5
datapnts_best5   embfunc3             pairpot23_full   potparas_best6
datapnts_best6   embfunc4             pairpot24        potparas_best7
datapnts_best7   fitdbse              pairpot24_full   potparas_best8
datapnts_best8   idsforsmallestsepns  pairpot33        potparas_best9
datapnts_best9   pairpot11            pairpot33_full   sepnHistogram.out
elecdens1        pairpot11_full       pairpot34        settings
elecdens1_full   pairpot12            pairpot34_full   smallestsepnvsstruc.dat
elecdens2        pairpot12_full       pairpot44
elecdens2_full   pairpot13            pairpot44_full
```

The relative 'goodness' of the R_{fit} values do not always depend strictly on the time that the program spends running (again, because of the random sampling) but generally the longer that the program runs, the better the R_{fit} values will become. If, like explained in section **??**, your values of R_{fit} do not begin to approach a useful value of R_{fit} 0.06 then consider searching for and removing very high energy atomic configurations because it seems that if there are many high energy configurations then MEAMfit will have trouble reaching a reasonable R_{fit} value independent of the run time.

Once the fitting has completed and converged to a sufficiently reasonable result, we can test the MEAM file by evaluating the optimization function for a testing set of energies. By the way that MEAMfit is set up, the best potential file will be named 'potparas_best1'. Run the following commands:

```
cp potparas_best1 potparas_MEAM
cp settings settings_original
vim settings
```

Copy and paste the following settings into the new settings file that you have just created using vim:

```
TYPE=MEAM
POTFILEIN=potparas_MEAM
CUTOFF_MAX=4.4
NTERMS=2
NTERMS_EMB=3
NOOPT=true
```

Close vim by pressing the escape key and then :wq followed by enter (for more coverage of vim, please see section 3.7). Run MEAMfit again to evaluate the potential against a testing set. Please see the following standard output for my running MEAMfit against a testing set for the potential I evaluated with 1575 vsprun.xml files that reached $R_{fit} = 7.659225012101019E-002$:

```
---------------- MEAMfit (version 1.02) ----------------
By Andrew I. Duff and Marcel H. F. Sluiter, 2006-2015
--------------------------------------------------------


 Initializing Fitting database
 -------------------------------
 File | Configs to fit | Quantity to fit | Weights
 vasprun1.xml 1-1 Free energy    1.00000000000000
 vasprun10.xml 1-1 Free energy    1.00000000000000
 vasprun100.xml 1-1 Free energy    1.00000000000000
 vasprun1000.xml 1-1 Free energy    1.00000000000000
```

I'm skipping some standard output lines here as they just list the large number of vasprun.xml files. Interesting parts of the standard output follow:

```
 vasprun995.xml 1-1 Free energy    1.00000000000000
 vasprun996.xml 1-1 Free energy    1.00000000000000
 vasprun997.xml 1-1 Free energy    1.00000000000000
 vasprun998.xml 1-1 Free energy    1.00000000000000
 vasprun999.xml 1-1 Free energy    1.00000000000000
 (fitting to  1575 atomic configurations across all files)

 General initialization
 -----------------------
 No SEED in settings file, using time to seed random numbers.
 No MUT_PROB in settings file, using default (0.3).
 No NOPTFUNCSTORE in settings file, using default (10).
 No OPTFUNCCG in settings file, using default (OPTFUNCCG=10).
 No OPTFUNC_ERR in settings file, using default for energy fit (OPTFUNC_ERR=10^-
14).
 No OPTFUNCCG_GA in settings file, using default (OPTFUNCCG=
 10*OPTFUNCCG 100.0000000000).
 No OPTDIFF in settings file, using default (=10^-10)
 No OPTACC in settings file, using default (=0.0005)
 No STOPTIME in settings file, using default (=168 hours=1 week)
 No MAXFUNCEVALS in settings file, using default (=2000)
 POTFILEIN=potparas_MEAM (from settings)
 No FIXPOTIN in settings file: allow input potential to optimize
 NOOPT=TRUE (from settings file)
 CUTOFF_MAX=   4.40000 (from settings file)
 No CUTOFF_MIN in settings file, using default (CUTOFF_MIN=
    1.50000000000000       )
 TYPE=MEAM (from settings file)

 Potential initialization
 -------------------------
 Reading in potential parameters from potparas_MEAM
 Completed structure initialization
 Preparing to read data from vasprun.xml files
 Completed data read-in from vasprun.xml files
 avgEn=  -68.8679761438603      , avgFrc=  0.000000000000000E+000
 varEn=   3200.92505263403      , varFrc=  0.000000000000000E+000

 ---------------------------------------------
 Optimization function=  0.100834984857534
 ---------------------------------------------

 rms error on energies=   5.70491253221082
 rms error on forces=  0.000000000000000E+000

 Energies:

 Structure           fitdata        truedata
 ------------------------------------------------
vasprun1.xml_1                 0.000000000000000000       0.000000000000000000
vasprun10.xml_1                1.646281788326703577      -0.180839689999999109
vasprun100.xml_1              64.398263877670515853      59.682770330000003867
vasprun1000.xml_1             -9.071981776729828084     -13.408281660000000102
vasprun1001.xml_1             -8.934956852083615786     -13.768509769999994319
```

I skip many lines from the standard output here since there are 1575 vasprun.xml files and this section lists the fit data vs the true data for every one of them, that's much too much to include here. For brevity, I just include the tail end of the optimization energies and all the standard output that follows:

```
vasprun995.xml_1             -10.191290211528070131     -11.579342849999989085
vasprun996.xml_1              -9.849236737160453004     -11.887335669999998800
vasprun997.xml_1              -9.605878160690579648     -12.249370020000000636
vasprun998.xml_1              -9.400435171991816219     -12.636695009999996842
vasprun999.xml_1              -9.224844218213647196     -13.027656729999989693

 ------------------------
 Optimization completed
 Total time taken 15s.
 ------------------------

 Camelion output:
```

```
------------------
Substitute the following lines into the goion script (note, reduced units):

set AlAl_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set AlO_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set AlN_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set AlTi_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set OO_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set ON_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set OTi_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set NN_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set NTi_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set TiTi_PV=(10000  1.63934426229508  0.268744961031981E-03 0.0 0.0 0.0)
set Al_EDV=(T 10000  1.63934426229508  0.268744961031981E-03 0.0)
set O_EDV=(T 10000  1.63934426229508  0.268744961031981E-03 0.0)
set N_EDV=(T 10000  1.63934426229508  0.268744961031981E-03 0.0)
set Ti_EDV=(T 10000  1.63934426229508  0.268744961031981E-03 0.0)
set Al_EB=(14000  96711.7988394584  6.90847909418233 -6.90847909418233)
set O_EB=(14000  96711.7988394584  6.90847909418233 -6.90847909418233)
set N_EB=(14000  96711.7988394584  6.90847909418233 -6.90847909418233)
set Ti_EB=(14000  96711.7988394584  6.90847909418233 -6.90847909418233)
set Al_TWH1=0.013392038853330
set Al_TWH2=-0.00106318765798
set Al_TWH3=40.14012998108759
set O_TWH1=6.9729513102705898
set O_TWH2=57.934329718367997
set O_TWH3=******************
set N_TWH1=-2.196854830390480
set N_TWH2=0.0005874172956921
set N_TWH3=0.0000885875701606
set Ti_TWH1=-13.4034321728207
set Ti_TWH2=-25.6340266491969
set Ti_TWH3=-0.39374397307126

!!! Also change the POTENTIALDIRECTORY= line to current directory !!!

Please also add appropriate values for:
set Al_MASS_FORCECONSTANT=(#num #num)
set O _MASS_FORCECONSTANT=(#num #num)
set N _MASS_FORCECONSTANT=(#num #num)
set Ti_MASS_FORCECONSTANT=(#num #num)
set OMEGA_Al=#num
set OMEGA_O=#num
set OMEGA_N=#num
set OMEGA_Ti=#num
(please see Camelion documentation for further details)
```

Searching through the beginning half of the standard output, we can find that:

```
Optimization function=  0.100834984857534
```

From the MEAMfit documentation, we know that R_{test} $R_{fit} + 0.015$ so then, in this instance, $R_{test} - R_{fit} = 0.0242427347$. You need to verify for yourself that the errors and the value of R_{test} vs. R_{fit} are reasonable for your own calculation(s).

The contents of the directory (minus all of the numerous vasprun.xml files) will look something similar to the following:

```
uname@uname:~/vasprun_files> ls
```

```
AlONTi_MEAM.eb     datapnts_best9          pairpot11        pairpot34_full
AlONTi_MEAM.pv     elecdens1               pairpot11_full   pairpot44
AlONTi_MEAM0.edv   elecdens1_full          pairpot12        pairpot44_full
AlONTi_MEAM1.edv   elecdens2               pairpot12_full   potparas_MEAM
AlONTi_MEAM2.edv   elecdens2_full          pairpot13        potparas_best1
AlONTi_MEAM3.edv   elecdens3               pairpot13_full   potparas_best10
MEAMfit.sb         elecdens3_full          pairpot14        potparas_best2
bestoptfuncs       elecdens4               pairpot14_full   potparas_best3
datapnts_best1     elecdens4_full          pairpot22        potparas_best4
datapnts_best10    embfunc1                pairpot22_full   potparas_best5
datapnts_best2     embfunc2                pairpot23        potparas_best6
datapnts_best3     embfunc3                pairpot23_full   potparas_best7
datapnts_best4     embfunc4                pairpot24        potparas_best8
datapnts_best5     fitdbse                 pairpot24_full   potparas_best9
datapnts_best6     fitted_quantities.out   pairpot33        sepnHistogram.out
datapnts_best7     idsforsmallestsepns     pairpot33_full   settings
datapnts_best8     largestrho.dat          pairpot34        smallestsepnvsstruc.dat
```

The same set of vasprun.xml files may not always reproduce the same results between different calculation runs of MEAMfit even with similar calculation times, this is a result again from the random sampling that MEAMfit uses as initial guesses for atomic configurations. For example, we can run MEAMfit again in a separate directory on the same set of vasprun.xml files and arrive at a different set of optimization functions:

```
 Top           10  optfuncs:
                1 :  5.186846000366864E-002  time:         45  hours
                2 :  8.378815249695254E-002  time:         37  hours
                3 :  8.598895680046785E-002  time:         31  hours
                4 :  8.642022938336989E-002  time:         41  hours
                5 :  9.708726029563895E-002  time:         14  hours
                6 :  0.107633313843299       time:         34  hours
                7 :  0.130808288358416       time:         18  hours
                8 :  0.134429701506718       time:          7  hours
                9 :  0.167181936878481       time:         26  hours
               10 :  0.197315169364415       time:         10  hours

 Total time taken    45h 12m  0s.
```

```
General initialization
-----------------------
 No SEED in settings file, using time to seed random numbers.
 No MUT_PROB in settings file, using default (0.3).
 No NOPTFUNCSTORE in settings file, using default (10).
 No OPTFUNCCG in settings file, using default (OPTFUNCCG=10).
 No OPTFUNC_ERR in settings file, using default for energy fit (OPTFUNC_ERR=10^-
 14).
 No OPTFUNCCG_GA in settings file, using default (OPTFUNCCG=
 10*OPTFUNCCG 100.0000000000).
 No OPTDIFF in settings file, using default (=10^-10)
 No OPTACC in settings file, using default (=0.0005)
 No STOPTIME in settings file, using default (=168 hours=1 week)
 No MAXFUNCEVALS in settings file, using default (=2000)
 POTFILEIN=potparas_MEAM (from settings)
 No FIXPOTIN in settings file: allow input potential to optimize
 NOOPT=TRUE (from settings file)
 CUTOFF_MAX=   4.40000 (from settings file)
 No CUTOFF_MIN in settings file, using default (CUTOFF_MIN=
   1.50000000000000      )
```

```
TYPE=MEAM (from settings file)

 Potential initialization
 ---------------------------
 Reading in potential parameters from potparas_MEAM
 Completed structure initialization
 Preparing to read data from vasprun.xml files
 Completed data read-in from vasprun.xml files
 avgEn=  -68.8679761438603       , avgFrc=  0.000000000000000E+000
 varEn=   3200.92505263403       , varFrc=  0.000000000000000E+000

 ----------------------------------------------
 Optimization function=  6.303620673965127E-002
 ----------------------------------------------

 rms error on energies=   3.56638171087305
 rms error on forces=  0.000000000000000E+000
```

In this case, we find that there is a much better fit function in the set of best optimization functions and that the difference between the fit and test values $R_{test} - R_{fit} = 0.011352$ which constitutes a substantially superior result in the fitting set without a change in the set of supplied vasprun.xml files.

It is recommended by the authors to run multiple parallel instances of MEAMfit all over reasonably long time scales in order to maximize your chances of attaining a useful value of R_{fit}.

10.7 Removing selected calculations from a set with automation

Sometimes you may desire to remove certain files from a large set of automatically-created VASP calculations. This may be because MEAMfit can have a hard time with very high energy values or because you are just attempting to reduce the intensity of a calculation that MEAMfit will need to perform. In either case, we can save a tremendous amount of hassle by using BASH to perform this recursively.

For the sake of this example, we will assume that you have a directory tree similar to the following figure (which is also given in section 10.4):

```
                              Top Directory:
                        +------------+     +-------------+
                        | Crystals |       | Molecules |
                        +------------+     +-------------+

                         Second Level Directories:
                      +-----+ +-------+ +------+      +---+ +---+
                      |TiN| |Al2O3| |TiO2|          |N2| |O2|
                      +-----+ +-------+ +------+      +---+ +---+

                          Third Level Directories:
                      +-------------------------------------+
                      |Individual Calculations 1...2...n|
                      +-------------------------------------+
```

And that you are currently located in one of the second level directories as shown in the figure above. If, for example, you have a calculation set that you want to trim some of the vasprun.xml files out of that all reside in the third level directories, then we can do this recursively with for loops in BASH very simply. For the sake of this example, let's assume that you have calculation sets of dimers approaching slabs with distances from the top of the slab to the first atom ranging from 0.05 to 3.00Å.

Because 0.05Å might be too close for a reasonable interaction, and 3.00Å might be too far away to be interesting, we can consider a means of cutting out the vasprun files in a certain range. For the sake of this demonstration, we will choose to cut out all of the vasprun.xml files that are less than 0.50Å and greater than 2.50Å in separation.

The command that follows (all done in one single line) will run two levels of for loops. In the first level of for loop, we create a variable called $a and loop that over every directory in the current directory. We navigate into directory $a and then run two additional for loops where we delete directories with parts of names matching a sequence that is given. In this case, the sequences correspond to names that I assigned during VASP calculations which include the interatomic separation (for more on how these VASP calculations were set up and run, see sections 9.4, and 9.5). Finally, we terminate the for loops which operate in all of the third level directories, list the contents of the directory, navigate up a level, enter the next directory, and do it all again until the conditions of the for loop are exhausted.

Please see the following terminal command:

```
for a in */; do cd $a ; Sequence1=$(seq -w 0.05 0.05 0.40); Sequence2=$(seq -w 2.50 0.05 3.00); for lat in ${Sequence1}; do rm -r *${lat} ; done; for lat in ${Sequence2}; do rm -r *${lat} ; done; ls; cd .. ; done
```

11 Shell Scripting for Force Field Creation with potfit

Potfit [17, 18, 19, 20] is a program capable of fitting a variety of interatomic potentials including EAM or MEAM potentials to a set of atomic configurations. With its built in tools, it can convert data from a variety of DFT output formats like castep and VASP into data that it can fit. Like with MEAMfit (see chapter 10 for more), this functionality is especially useful in the case that you cannot find a suitable interatomic potential to accurately describe the system that you are considering or if you want to develop a custom force field for your own purposes. These force fields are directly applicable in molecular dynamics calculations, especially with programs like Sandia National Lab's LAMMPS.

11.1 Compiling potfit

Potfit is straightforward to compile and uses the Python build automation tool WAF to do all of the heavy lifting for you. Please see a terminal transcript of the compilation and checking the compiled executable for potfit-20210702. In this case, we are compiling potfit for the generation of MEAM force fields:

```
uname@uname:~/codes/> wget https://www.potfit.net/download/potfit-20210702.tar.gz
--2021-10-28 16:34:12--  https://www.potfit.net/download/potfit-20210702.tar.gz
Resolving www.potfit.net (www.potfit.net)... 2a03:4000:1d:345::1, 185.183.158.219
Connecting to www.potfit.net (www.potfit.net)|2a03:4000:1d:345::1|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 395930 (387K) [application/x-gzip]
Saving to: 'potfit-20210702.tar.gz'

potfit-20210702.tar.gz
    100%[=============================================>] 386.65K  624KB/s   in 0.6s

2021-10-28 16:34:14 (624 KB/s) - 'potfit-20210702.tar.gz' saved [395930/395930]

uname@uname:~/codes/> untar potfit-20210702.tar.gz
uname@uname:~/codes> cd potfit/
uname@uname:~/codes/potfit> ls
CHANGELOG  CONTRIBUTORS  LICENSE  Makefile  Makefile.inc  bin  build  examples  src
     tests  util  waf  wscript
uname@uname:~/codes/potfit> ./waf --help
waf [commands] [options]

Main commands (example: ./waf build -j4)
  build     : executes the build
  clean     : cleans the project
  configure: configures the project
  dist      : makes a tarball for redistributing the sources
  distcheck: checks if the project compiles (tarball from 'dist')
  install   : installs the targets on the system
  list      : lists the targets to execute
  step      : executes tasks in a step-by-step fashion, for debugging
  uninstall: removes the targets installed

Options:
  --version             show program's version number and exit
  -c COLORS, --color=COLORS
```

```
                        whether to use colors (yes/no/auto) [default: auto]
  -j JOBS, --jobs=JOBS  amount of parallel jobs (64)
  -k, --keep            continue despite errors (-kk to try harder)
  -v, --verbose         verbosity level -v -vv or -vvv [default: 0]
  --zones=ZONES         debugging zones (task_gen, deps, tasks, etc)
  -h, --help            show this help message and exit

  Configuration options:
    -o OUT, --out=OUT   build dir for the project
    -t TOP, --top=TOP   src dir for the project
    --prefix=PREFIX     installation prefix [default: '/usr/local/']
    --bindir=BINDIR     bindir
    --libdir=LIBDIR     libdir
    --check-c-compiler=CHECK_C_COMPILER
                        list of C compilers to try [gcc clang icc]

  Build and installation options:
    -p, --progress      -p: progress bar; -pp: ide output
    --targets=TARGETS   task generators, e.g. "target1,target2"

  Step options:
    --files=FILES       files to process, by regexp, e.g. "*/main.c,*/test/main.o"

  Installation and uninstallation options:
    --destdir=DESTDIR   installation root [default: '']
    -f, --force         force file installation
    --distcheck-args=ARGS
                        arguments to pass to distcheck

  potfit general options:
    Please check the explanations on the potfit homepage for more details.
    --enable-bindist    Write a binned radial distribution file
    --enable-contrib    Enable support for box of contributing particles
    --enable-dsf        Use damped shifted force approach (coulomb-based
    interactions only)
    --enable-evo        Use evolutionary algorithm instead of simulated annealing
    --enable-fweight    Use modified weights for the forces
    --enable-mpi        Enable MPI parallelization
    --enable-nopunish   Disable punishments
    --enable-resc       Enable rescaling (use with care!)
    --enable-stress     Include stress in fitting process
    --enable-uq         Generate potential ensemble for uncertainty quantification

  potfit potential options:
    available interactions in alphabetical order are:
            adp             angular dependent potentials
            ang             angular pair potentials
            ang_elstat      angular pair potentials with eletrostatics
            coulomb         coulomb interactions
            dipole          dipole interactions
            eam             embedded atom method
            eam_coulomb     embedded atom method with coulomb interactions
            eam_dipole      embedded atom method with dipole interactions
            meam            modified embedded atom method
            pair            pair potentials
            stiweb          Stillinger-Weber potentials
            tbeam           two-band embedded atom method
            tersoff         Tersoff potentials
            tersoffmod      modified Tersoff potentials
    -i INTERACTION, --interaction=INTERACTION
                        one of the interactions listed above
    -m MODEL, --model=MODEL
                        support analytic, kim or tabulated potentials

  potfit math library options:
    available math libraries are:
            lapack          Linear Algebra PACKage from netlib.org
            mkl             Intel Math Kernel Library
    -l MATHLIB, --math-lib=MATHLIB
                        Select math library to use (default: mkl)
    --math-lib-base-dir=MATH_LIB_BASE_DIR
                        Base directory of selected math library

  potfit debugging options:
    --debug             Build binary with debug information
    --asan              Build binary with address sanitizer support
    --profile           Build binary with profiling support
waf: error: no such option: -e
uname@uname:~/codes/potfit> ./waf configure -i meam -m apot
Setting top to                           : ~codes/potfit
Setting out to                           : ~codes/potfit/build
Checking for 'gcc' (C compiler)          : /usr/bin/gcc
Checking for header mkl_vml.h            : yes
Checking for header mkl_lapack.h         : yes
Compiling MKL test binary                : OK

potfit has been configured with the following options:
potential model      = apot
interaction          = meam
math library         = mkl

Now run './waf' to start building potfit

'configure' finished successfully (1.274s)
uname@uname:~/codes/potfit> ./waf
Waf: Entering directory `~codes/potfit/build'
[ 1/30] Compiling src/functions_impl.c
[ 2/30] Compiling src/functions.c
[ 3/30] Compiling src/potential_output_imd.c
[ 4/30] Compiling src/mpi_utils.c
[ 5/30] Compiling src/config.c
[ 6/30] Compiling src/potential_output.c
[ 7/30] Compiling src/potential_input_f5.c
[ 8/30] Compiling src/memory.c
[ 9/30] Compiling src/potential_input_f4.c
[10/30] Compiling src/potential_input_f3.c
[11/30] Compiling src/errors.c
[12/30] Compiling src/utils.c
[13/30] Compiling src/potential_input.c
[14/30] Compiling src/elements.c
[15/30] Compiling src/random_dsfmt.c
[16/30] Compiling src/params.c
[17/30] Compiling src/potential_output_lammps.c
[18/30] Compiling src/splines.c
[19/30] Compiling src/potential_input_f0.c
[20/30] Compiling src/random.c
```

```
[21/30] Compiling src/force_meam.c
[22/30] Compiling src/force_common.c
[23/30] Compiling src/simann.c
[24/30] Compiling src/optimize.c
[25/30] Compiling src/linmin.c
[26/30] Compiling src/brent.c
[27/30] Compiling src/powell_lsq.c
[28/30] Compiling src/bracket.c
[29/30] Compiling src/potfit.c
[30/30] Linking build/src/potfit_apot_meam_mkl
Waf: Leaving directory `~codes/potfit/build'

---> Successfully moved potfit_apot_meam_mkl to bin/ folder <---

'build' finished successfully (1.052s)
uname@uname:~/codes/potfit>

uname@uname:~/codes/potfit> cd bin/
uname@uname:~/codes/potfit/bin> ls
potfit_apot_meam_mkl
uname@uname:~/codes/potfit/bin> file potfit_apot_meam_mkl
potfit_apot_meam_mkl: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
    dynamically linked, interpreter /lib64/l, BuildID[sha1]=83
    df7d4e482de450bec65998985e8ec870ef1a1c, for GNU/Linux 3.2.0, with debug_info,
    not stripped
uname@uname:~/codes/potfit/bin>

uname@uname:~/codes/potfit/bin> ./potfit_apot_meam_mkl
This is potfit-20210702 (7e5bf091) compiled on Oct 28 2021, 16:17:47.

[ERROR] Usage: ./potfit_apot_meam_mkl <paramfile>
uname@uname:~/codes/potfit/bin>
```

11.2 Collecting many VASP OUTCAR files for potfit

Similar to MEAMfit (see chapter 10 for more on MEAMfit and MEAMfit2), potfit can use the output of VASP calculations to generate interatomic potentials. Instead of the vasprun.xml files that MEAMfit uses, potfit uses VASP's OUTCAR files instead. In the case that you have numerous atomic configurations with numerous OUTCAR files each, it is intelligent to automate the collection of OUTCAR files.

As we discussed in section 10.4, we will assume here that all of the VASP calculations from which you want to collect the OUTCAR files are arranged as is shown in the below diagram. Here, sets of calculations are grouped together by general class, then, in a sub-directory, their specific structure, and then, in a further sub-directory, the individual atomic configuration calculations:

```
                          Top Directory:
                    +----------+     +-----------+
                    | Crystals |     | Molecules |
                    +----------+     +-----------+

                      Second Level Directories:
                 +---+ +-----+ +----+      +--+ +--+
                 |TiN| |Al2O3| |TiO2|      |N2| |O2|
                 +---+ +-----+ +----+      +--+ +--+

                              Third Level Directories:
                           +------------------------------------+
                           |Individual Calculations 1...2...n|
                           +------------------------------------+
```

The following script can be used to collect OUTCAR files. The script uses three nested for loops to do the heavy lifting as well as a variable named $File_Number which is updated at each step in the for loops that appends a unique identifier number to each of the collected OUTCAR files so as to not cause issues with ambiguity. This script is very similar to one supplied in section 10.4 of this text, demonstrating the versatility of these scripts and how they can be mixed and matched to suit your various needs.

Please see the following script for the collection of OUTCAR files:

```
mkdir OUTCAR_files
File_Number="1" # Begin a variable to use in the for() loop to count the current job
     up to the final job

for a in */ ; do # Loop within top directory e.g., Crystals  Molecules  Slabs
  echo "Entering directory $a"
  cd "$a" ; pwd

    for b in */ ; do # Loop within structure directory e.g., Al2O3, Ti2O, etc...
      echo "Entering directory $b"
      cd "$b" ; pwd

          for c in */ ; do # Loop within a single calculation directory e.g.,
    Al2O3_a0_0.980, Al2O3_a0_0.981, etc...
            echo "Entering directory $c"
            cd "$c" ; pwd

              cp OUTCAR ../../../OUTCAR_files/OUTCAR${File_Number} # Copy and rename
     the OUTCAR file to the target directory based on the current File_Number
              echo " Copied OUTCAR to OUTCAR${File_Number}"
              File_Number=$(echo `expr $File_Number + 1`)  # Add 1 $File_Number

            cd ..
          done # End of single calculation directory for() loop

      cd ..
    done # End of major calculation category directory for() loop

  cd ..
done  # End of main for() loop   # Iterate until for loop conditions are exhausted

echo " OUTCAR file collection and renaming completed"
echo " Success, End of Script"
```

11.3 Using the potfit vasp2force built-in

Used in concert with the OUTCAR collector script covered in section 11.2, the vasp2force script supplied with potfit can be used to generate a data set compatible with potfit. The vasp2force script is located in the util directory of your potfit install as can be seen here:

```
uname@uname:~/codes/potfit/util> ls
README  castep2force  devel  force2imd  force2poscar  kim  list_config  logo
    makeapot  old  plotapot  potfit  potfit_setup  vasp2force
```

```
uname@uname:~/codes/potfit/util> ./vasp2force
Searching directory . for OUTCAR* files ...
Could not find any OUTCAR files in this directory.
```

Running this code can be very straightforward as we will demonstrate in the following on just a single VASP OUTCAR file calculated for the Al2O3 system:

```
uname@uname:~/test-dir> ls
OUTCAR
uname@uname:~/test-dir> ~/codes/potfit/util/vasp2force -h
usage: vasp2force [-h] [-c <elem_list>] [-e <sae_file>] [-a] [-f] [-l] [-r]
                  [-s CONFIGS] [-w WEIGHT]
                  [files [files ...]]

Converts vasp output data into potfit reference configurations.

positional arguments:
  files                 list of OUTCAR files (plain or gzipped)
                        (directory in case of -r option)

optional arguments:
  -h, --help            show this help message and exit
  -c <elem_list>        list of indices for chemical elements to use
                        e.g. -c Mg=0,Zn=1
  -e <sae_file>         file with single atom energies
  -a, --all             use all configurations from OUTCAR
  -f, --final           use only the final configuration from OUTCAR
  -l, --list            list OUTCAR properties and exit
  -r, --recursive       scan recursively for OUTCAR files
  -s CONFIGS, --configs CONFIGS
                        comma separated list of configurations to use
                        supported schemes:
                         -s 1,4,12          use configs 1, 4 and 12
                         -s 1,6-9           use configs 1, 6, 7, 8 and 9
                         -s 1,4,6-9,12      combination of the above
  -w WEIGHT, --weight WEIGHT
                        configuration weight for all configurations
uname@uname:~/test-dir> ~/codes/potfit/util/vasp2force -f
Searching directory . for OUTCAR* files ...
Found the following files:
  ./OUTCAR
#N 10 1
#C Al O
## force file generated from file ./OUTCAR config 1
#X     5.17795526     0.00000000     0.00000000
#Y     2.94847555     4.25649065     0.00000000
#Z     2.94847555     1.54436294     3.96644119
#W 1.000000
#E -7.4885214460
#S -0.0167987 -0.01679376 -0.01679276 -3.564295e-06 -1.27965e-06 -2.440699e-06
#F
0      1.63802      0.85797      0.58665    -0.012224    -0.006403    -0.004378
0      3.89943      2.04246      1.39657     0.012224     0.006403     0.004378
0      7.17548       3.7584      2.56987    -0.012224    -0.006403    -0.004378
0      9.43688      4.94288      3.37979     0.012224     0.006403     0.004378
1      7.26414      2.75332      0.99161    -0.007678     0.014658           -0
1      4.67516      3.27908       0.2227    -0.007678     0.005318     0.013659
1      6.39975      2.52178      3.74374     0.007678    -0.005318    -0.013659
1       5.7172       4.8764      2.20592            0     -0.00934     0.013659
```

```
1       5.3577      0.92446     1.76052            0      0.00934    -0.013659
1      3.81077      3.04753     2.97483     0.007678    -0.014658            0
uname@uname:~/test-dir>
```

This is equivalent to what the potfit documentation calls a simple configuration file (e.g., one containing a single atomic configuration). As of the time of this writing, the vasp2force executable will not write this configuration file to your disk, instead it will only print the configuration to the terminal. Since we will need to have this saved as a configuration file, you can run the following command to dump that terminal output into a file of your choosing (see section 4.2 for more on this specific functionality):

```
vasp2force -f |& tee -a vasp2force.out
```

Combining the tee command with executing the vasp2force program lets you create a textfile containing all of the potfit-formatted atomic configurations. If we then combine this functionality with the script to collect many VASP OUTCAR files given in section 11.2, then we can create potfit-compatible configuration files of many atomic configurations. All you need to do is run your vasp2force executable inside of the directory that you've collected all of your OUTCAR files into. The tee command here will collect all of the terminal output however so you will need to trim the first n lines of vasp2force.out file that we've created to remove all of the lines saying something to the effect of the following:

```
  ./OUTCAR1174
  ./OUTCAR1132
  ./OUTCAR112
  ./OUTCAR365
  ./OUTCAR646
  ./OUTCAR829
  ./OUTCAR641
  ./OUTCAR590
  ./OUTCAR882
  ./OUTCAR314
  ./OUTCAR427
  ./OUTCAR36
```

This header is followed by n of these ./OUTCARn lines which can be removed using the shell. First, find the number of files that are in the current directory (in which we are running the vasp2force executable with the following command:

```
uname@uname:~/test-dir> ls | wc -l
1228
```

This command pipes a list of the file contents into the word count built-in (wc) and uses the lines (-l) option to count the number of files inside of the directory. In this specific case, the number of files is 1228, so we will have to remove the first 1228 lines from the vasp2force.out file. Let's take a quick look at the vasp2force.out file just to make sure that we're doing everything we need to do to clean up the file:

```
uname@uname:~/test-dir> head vasp2force.out
Searching directory . for OUTCAR* files ...
Found the following files:
  ./OUTCAR1174
  ./OUTCAR1132
  ./OUTCAR112
  ./OUTCAR365
  ./OUTCAR646
```

```
./OUTCAR829
./OUTCAR641
./OUTCAR590
```

Notice that the vasp2force executable has added an additional two lines to the head of the file! So then we'll have to add two to the n files that we counted using the word count command from before and then remove that total number of lines from the head of the file. This can be done using sed in the following way:

```
uname@uname:~/test-dir> sed -i -e '1,1230d' vasp2force.out
uname@uname:~/test-dir> head vasp2force.out
#N 10 1
#C N Ti
## force file generated from file ./OUTCAR1174 config 1
#X     4.23500013     0.00000000     0.00000000
#Y     0.00000000     4.23500013     0.00000000
#Z     0.00000000     0.00000000    15.00000000
#W 1.000000
#E -8.7514170700
#S -0.04021155 -0.04021155 0.01776617         0         0         0
#F

```

I'll leave it as an exercise to the reader to automate this all into a single script, it may be helpful to refer to the expr command discussed at more length in chapter 9 to handle the addition of integers to a variable (as you would need to do to add two to the n OUTCAR files that you counted with the wc command. In either case however, it's pretty quick to just run the wc and sed commands in the terminal without a script.

11.4 Using the potfit makeapot built-in

The potfit program called makeapot is used to make an initial analytic potential file based on the interactions that you're interested in before the optimization begins. The initial potential file is required for the configuration file and so we will cover its creation briefly here.

In my build of potfit, for some reason, the makeapot executable was empty after the build. To fix this, I just copied and pasted the code from another download of potfit into the executable using vim which fixed the problem. Creating the potential file with makeapot requires knowledge of the number of atom types. Based on the number of atom types, there will be different required numbers and types of potential functions. In the case of a MEAM potential with 4 distinct species, there are 32 required functions: ten for the electrostatic core-core repulsion $\Phi_{ij}(r)$, four for the electron transfer $\rho_j(r)$, four for the energy describing an ion's core as it is embedded in the electron sea $F_i(n)$, ten for the three-body terms $f_{ij}(r)$, and four more terms for the three-body angular function $g_i(cos\theta)$. For different numbers of atomic species, the number of functions will change.

We can use the makeapot executable to create a meam potential that potfit can start with by using the following commands. Here we use the tee and pipe commands (for more on tee and this functionality, see section 4.2) as well to write the potential to a file called meam.pot:

```
uname@uname:~/test-dir> ~/codes/potfit/util/makeapot -h
usage: makeapot [-h] [-n NTYPES] [-c CUTOFF] [-g] [-r] -i INTERACTION [-l]
                [--cp] [-f FUNCTIONS] [-o OUTFILE] [-e ELEMENTS]

Create an analytic potential file for potift.

```

```
optional arguments:
  -h, --help        show this help message and exit
  -n NTYPES         number of atoms types, runs from 0 to N-1
  -c CUTOFF         cutoff radius (default 6.0)
  -g                use a global cutoff parameter for all potentials
  -r                randomize the values for the potential parameters
  -i INTERACTION    supported interaction types are: adp, eam, meam, pair,
                    stiweb, tersoff
  -l, --list        list options which are available
  --cp              enable chemical potentials (only for pair)
  -f FUNCTIONS      comma separated list of potential functions, either name or
                    i*name, where i=1,2,3,...
  -o OUTFILE        write output to this file instead of stdout
  -e ELEMENTS       comma separated list of elements for #C header line

To specify multiple potentials you can use the following syntax:

  makeapot -n 3 -i eam -f 6*eopp,3*csw,3*bjs

which uses 6 eopp potentials, 3 csw and 3 bjs in this order.

uname@uname:~/test-dir> ~/codes/potfit/util/makeapot -n 4 -i meam -f 10*eopp_sc,4*
    csw2_sc,4*bjs,10*csw2_sc,4*parabola |& tee -a meam.pot
#F 0 32
#T MEAM
#I 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#E

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 -100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 -100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 -100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

```

```
type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 −100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 −100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 −100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 −100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 −100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
h             1.00 0.50 2.00

type eopp_sc
cutoff 6.0
C_1           15.00 1.00 10000.00
eta_1         6.00 1.00 20.00
C_2           5.00 −100.00 100.00
eta_2         3.00 1.00 10.00
k             2.50 0.00 6.00
phi           3.00 0.00 6.30
```

```
h              1.00  0.50  2.00

type eopp_sc
cutoff 6.0
C_1            15.00  1.00  10000.00
eta_1          6.00  1.00  20.00
C_2            5.00  −100.00  100.00
eta_2          3.00  1.00  10.00
k              2.50  0.00  6.00
phi            3.00  0.00  6.30
h              1.00  0.50  2.00

type csw2_sc
cutoff 6.0
a              0.20  −2.00  2.00
alpha          2.00  1.00  6.00
phi            0.00  0.00  6.30
beta           3.00  0.50  5.00
h              1.00  0.50  2.00

type csw2_sc
cutoff 6.0
a              0.20  −2.00  2.00
alpha          2.00  1.00  6.00
phi            0.00  0.00  6.30
beta           3.00  0.50  5.00
h              1.00  0.50  2.00

type csw2_sc
cutoff 6.0
a              0.20  −2.00  2.00
alpha          2.00  1.00  6.00
phi            0.00  0.00  6.30
beta           3.00  0.50  5.00
h              1.00  0.50  2.00

type csw2_sc
cutoff 6.0
a              0.20  −2.00  2.00
alpha          2.00  1.00  6.00
phi            0.00  0.00  6.30
beta           3.00  0.50  5.00
h              1.00  0.50  2.00

type bjs
cutoff 6.0
F_0            −1.00  −10.00  0.00
gamma          2.00  0.10  2.00
F_1            2.00  1.00  5.00

type bjs
cutoff 6.0
F_0            −1.00  −10.00  0.00
gamma          2.00  0.10  2.00
F_1            2.00  1.00  5.00

type bjs
cutoff 6.0
```

```
F_0          −1.00 −10.00 0.00
gamma        2.00 0.10 2.00
F_1          2.00 1.00 5.00

type bjs
cutoff 6.0
F_0          −1.00 −10.00 0.00
gamma        2.00 0.10 2.00
F_1          2.00 1.00 5.00

type csw2_sc
cutoff 6.0
a            0.20 −2.00 2.00
alpha        2.00 1.00 6.00
phi          0.00 0.00 6.30
beta         3.00 0.50 5.00
h            1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a            0.20 −2.00 2.00
alpha        2.00 1.00 6.00
phi          0.00 0.00 6.30
beta         3.00 0.50 5.00
h            1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a            0.20 −2.00 2.00
alpha        2.00 1.00 6.00
phi          0.00 0.00 6.30
beta         3.00 0.50 5.00
h            1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a            0.20 −2.00 2.00
alpha        2.00 1.00 6.00
phi          0.00 0.00 6.30
beta         3.00 0.50 5.00
h            1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a            0.20 −2.00 2.00
alpha        2.00 1.00 6.00
phi          0.00 0.00 6.30
beta         3.00 0.50 5.00
h            1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a            0.20 −2.00 2.00
alpha        2.00 1.00 6.00
phi          0.00 0.00 6.30
beta         3.00 0.50 5.00
h            1.00 0.50 2.00

```

```
type csw2_sc
cutoff 6.0
a             0.20 −2.00 2.00
alpha         2.00 1.00 6.00
phi           0.00 0.00 6.30
beta          3.00 0.50 5.00
h             1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a             0.20 −2.00 2.00
alpha         2.00 1.00 6.00
phi           0.00 0.00 6.30
beta          3.00 0.50 5.00
h             1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a             0.20 −2.00 2.00
alpha         2.00 1.00 6.00
phi           0.00 0.00 6.30
beta          3.00 0.50 5.00
h             1.00 0.50 2.00

type csw2_sc
cutoff 6.0
a             0.20 −2.00 2.00
alpha         2.00 1.00 6.00
phi           0.00 0.00 6.30
beta          3.00 0.50 5.00
h             1.00 0.50 2.00

type parabola
cutoff 6.0
alpha         1.00 −10.00 10.00
beta          1.00 −10.00 10.00
gamma         1.00 −10.00 10.00

type parabola
cutoff 6.0
alpha         1.00 −10.00 10.00
beta          1.00 −10.00 10.00
gamma         1.00 −10.00 10.00

type parabola
cutoff 6.0
alpha         1.00 −10.00 10.00
beta          1.00 −10.00 10.00
gamma         1.00 −10.00 10.00

type parabola
cutoff 6.0
alpha         1.00 −10.00 10.00
beta          1.00 −10.00 10.00
gamma         1.00 −10.00 10.00
```

This potential is now ready to be used in potfit.

11.5 Using the potfit potfit_setup built-in

To run potfit, we need to give it a parameter file which will point it to required information like files and names of headers, etc... These actually can be made by hand simply enough but potfit comes supplied with a utility (potfit_setup) that automates this process to some extent.

To run the potfit_setup executable, you'll need to give a configuration file, a potential file, and an output prefix for the run. Please see the following terminal output as an example:

```
uname@uname:~/test-dir> ~/codes/potfit/util/potfit_setup -h
usage: potfit_setup [-h] [-c config file] [-p potential file] [-s prefix]

Create a simple potfit parameter file from scratch.

optional arguments:
  -h, --help          show this help message and exit
  -c config file      name of the potfit configuration file
  -p potential file   name of the potfit potential file
  -s prefix           prefix for all files

The prefix takes precedence over the -c and -p switches. If the <prefix>.pot
and <prefix>.config files are not found, the values of -c and -p are checked.
uname@uname:~/test-dir> ~/codes/potfit/util/potfit_setup -c vasp2force.config -p
    meam.pot -s AlOTiN |& tee -a meam.param
ntypes     4
config     vasp2force.config
startpot   meam.pot
endpot     meam.pot_end
tempfile   AlOTiN.tmp

imdpot     AlOTiN.imd
plotfile   AlOTiN.plot
flagfile   STOP

write_pair   1
write_lammps   1
plotmin    0.1

imdpotsteps 5000
output_prefix AlOTiN

opt    1
anneal_temp auto
eng_weight   100
stress_weight 10
seed     42
apot_punish 0
```

This parameter file is now ready to use with potfit.

11.6 Running potfit to generate a MEAM potential

Using all of the previously created files in sections 11.3, 11.4, and 11.5, we can finally run potfit on the configuration file that we generated in section 11.5 in order to generate a MEAM potential.

Make certain that all of the atomic configurations that you are attempting to run with potfit have enough atoms to make potfit happy (if they do not, potfit will halt in the import step and ask you to remove the configuration(s) with too few atoms). Please see the following terminal session as an example for running potfit:

```
uname@uname:~/test-dir> ~/codes/potfit/bin/potfit_apot_meam_mkl meam.param
This is potfit-20210702 (7e5bf091) compiled on Oct 28 2021, 16:17:47.

Reading parameter file >> meam.param << ...
[WARNING] Unknown tag <stress_weight> in parameter file ignored!
Reading parameter file >> meam.param << ... done
Starting to read the potential file:
 - Potential file format 0 detected: analytic potentials
 - Using 32 MEAM potential(s) to calculate forces
 - Successfully read 32 potential table(s)
Reading potential file >> meam.pot << ... done
Reading configuration file >> vasp2force.config << and calculating neighbor lists
    ...
```

Inbetween these two sections of terminal output there may be warnings about the box size of the calculation with respect to the cutoff distance, unrecognized text that potfit will ignore, or errors regarding there being too few atoms. After all of those warnings, potfit will begin its fitting routine as shown below:

```
Reading configuration file >> vasp2force.config << and calculating neighbor lists
    ... done

Read 964 configurations (964 with forces, 0 with stresses)
with a total of 11884 atoms (3611 N (30.39%), 3240 Ti (27.26%), 1963 Al (16.52%),
    3070 O (25.83%)).

Minimal Distances Matrix:
Atom          N          Ti          Al          O   with
N 0.950000   1.400000   1.150000   1.150000
Ti   1.400000   2.538465   1.745822   1.150000
Al   1.150000   1.745822   1.683032   1.150000
O 1.150000   1.150000   1.150000   1.233811

Global energy weight: 100.000000

Starting optimization with 164 parameters.
Determining optimal starting temperature T ...
Performed 1640 trial steps, 832 of them were downhill.
Setting T=19783009.849117

  k T            m F            F_opt
  0 19783009.849117    0 36600606.854105  36600606.854105
```

Depending on your computational resources and your system complexity, potfit may take a large amount of time to reach a converged potential file. For reference, it may take 48-168 hours for MEAMfit (discussed in chapter 10) to reach a reasonably converged solution.

In the case that potfit initiates with a larger than realistic temperature, you can also set the temperature manually by changing the anneal_temp option in the parameter file from auto to some value like 10 (given commonly as the anneal temp in the potfit documentation's example files). In this case, potfit will skip the "Determining optimal starting temperature T ..." and jump to the optimization step like shown in the following:

```
Reading configuration file >> vasp2force.config << and calculating neighbor lists
    ... done
```

```

Read 964 configurations (964 with forces, 0 with stresses)
with a total of 11884 atoms (3611 N (30.39%), 3240 Ti (27.26%), 1963 Al (16.52%),
    3070 O (25.83%)).

Minimal Distances Matrix:
Atom          N          Ti          Al          O   with
N 0.950000   1.400000   1.150000   1.150000
Ti   1.400000   2.538465   1.745822   1.150000
Al   1.150000   1.745822   1.683032   1.150000
O 1.150000   1.150000   1.150000   1.233811

Global energy weight: 100.000000

Starting optimization with 164 parameters.
  k T            m F             F_opt
  0 10.000000    0 36600606.854105 36600606.854105
  0 10.000000    1 3886268.043080   3886268.043080
  0 10.000000    2 904743.127530 904743.127530
```

12 Additional Topics Just for Fun

These contents are just as the title suggests, cool stuff in the shell that I thought would be fun to include. Enjoy!

12.1 The fork bomb: a denial of service 'virus'

A fork bomb is a cute little piece of code that can relatively easily crash a system. A word of caution though, if you intend to run this code without any sort of safeguards, then do it on a system that you're not afraid to crash. The fork bomb appears to be very simple, as is shown below (not my own creation):

```
:(){ :|:& };:
```

Another implementation of this code is as follows where the only thing that has changed (beyond the more readable structuring of the code) is that the colon has been replaced by the word fork (again, not my own creation overall).

```
fork() {
    fork | fork &
}
fork
```

The idea of the fork bomb is that, while it looks simple, it is a primitive sort of virus that creates continuously replicating instances of itself which will starve the system of resources to the point of slowing or crashing. This is, in other words, a type of denial of service attack. Wikipedia (wikipedia.org/wiki/Fork_bomb) contains a very rigorous but plain explanation of how the fork bomb works: "In [the fork bomb], a function is defined (fork()) as calling itself (fork), then piping (|) its result to a background job of itself (&)."

So then that cute looking piece of code from before has some teeth. Overall, put a different way, :() is a function all of whose input is what is returned being piped directly back into the function and split at each step using the fork command. The fork bomb is a neat and tidy tiny chunk of code with strange capabilities!

12.2 Text diagrams for inline human-readable descriptions of your code

Sometimes you may need to run a large number of calculations that are very similar but distinct and having only a few key differences from one script to another. In this case, it is the obvious idea to include a detailed description line in the text of the script as a nice comment to yourself so that you can remember exactly what the script's intention is instead of having to scroll through what my be many lines of code in order to remember what that specific script was intended to calculate. While that sounds great in theory, I find it to be massively taxing on my eyes, especially after spending a large number of hours staring at small text in a terminal.

To somewhat remedy this situation, I like to made ASCII text based graphics or viewgraphs within my code at key points and/or the head of the document so that I can look at a glance and remember what the script was intended to calculate specifically. The graphics can be made manually or with a number of utilities that you can search for on the internet. Particularly, I find this to be a lifesaver when I'm trying to generate EAM or MEAM potential files from a large number of VASP calculation sets.

Included below are some examples of little viewgraphs which I have included in various calculation scripts while generating a force field file for use with LAMMPS. In the scripts, I was running many individual calculations with slightly changed interatomic distances per script and each of the 100 individual scripts were all on different systems or different system configurations. Managing all of those highly similar scripts became tedious and a pain in the eyes so I resorted to including some graphics with text like follows:

The viewgraph below shows an Al-O dimer (as it might form in the simplest case in a sputtering system with dimer length calculated by VASP) as it is approaching a (100)-oriented slab of TiN where the Al atom in the dimer is approaching a Ti atom in the slab and the O atom in the dimer is approaching an N atom in the slab. This was important when doing many such similar calculations on minutely varying system configurations.

```
#                +-------+     +-------+
#                |  Al   |<--->|   O   |   Al-O Dimer (1.6337A)
#                +-------+     +-------+ |
#                                        | Variable z-coord.
#                                        |
#                +-------+     +-------+ v   +-------+
#                |  Ti   |<--->|   N   |<--->|  Ti   |  TiN (100) slab surface
#                +-------+     +-------+     +-------+
```

The viewgraph below shows a general reminder of how a POSCAR file was set up having a variable that is set in the scripts called lat_a which controls the height of a dimer above a crystal slab in a VASP calculation.

```
#           Dimer Equilibrium Bond Length
#                +-------+     +-------+
#                |Atom_1|<--->|Atom_2|  |
#                +-------+     +-------+ | Variable z-coord. = lat_a
#                                        |
#                +-----------------------+ v
#                |     Slab Surface      |
#                +-----------------------+
```

The viewgraph below shows a reminder of what the various variables included in a POSCAR file are. I find this extremely useful when referring back to old scripts which could otherwise take excess time to remember exactly what was going on. The name of the game here is entirely to waste less time on trying to remember what's going on in a script (with the added benefit of maybe reducing fatigue from staring at tiny text in a terminal too).

```
#    +-------+
#    |Atom_1| ^ Dimer Atom 1 <--- Ti in this case is pointing up
#    +-------+ |
#              |lat_b=lat_a + dimer equilibrium bond length
#    +-------+ |
#    |Atom_2| v Dimer Atom 2 <--- N in this case is pointing down
#    +-------+ |
#              |lat_a=Surface z-coord. + variable z-coord.
#    +-------+ |
#    |Atom_3| v Slab Surface
#    +-------+
```

The viewgraph below shows a recap that I put at the head of some scripts where a vertically oriented Al-O dimer molecule is approaching an O atom on a crystalline slab of c-axis oriented sapphire. This is useful when you have many configurations of a system and want to view or remember some of the parameters or a general indication of the intention of the calculation.

```
#                               +--------+
#                               |   Al   | ^   Dimer Atom 1
#                               +--------+ |
#                                          |   (1.6337A)
#                               +--------+ |
#                               |   O    | <   Dimer Atom 2
#                               +--------+ |
#                                          |   Variable z-coord.
#                               +--------+ |
#                               |   O    | v   C- Sapphire Slab Surface
#                               +--------+
```

The viewgraph below shows a general header I include in calculations where I'm determining the minimum energy configuration of a structure (in this case the structure was F-43m Ti2Al) as the script generates an equally spaced number of calculation scripts that expand and contract the lattice parameter by 2 percent.

```
#                                                           +------------+
#                                       +------------+      |            |
#                       +--------+      |            |      |            |
#                       |        |      |   Ti2Al    |      |            |
#                       |        | <--- |   F-43m    | ---> |            |
#                       +--------+      +------------+      +------------+
#
#                    2% Contracted        Equilibrium         2% Expanded
```

The viewgraph below I thought was handy to include after several lines of code that use the bash program called bc to calculate a maximum and minimum lattice parameter over which to sweep a VASP energy calculation with inputs like a starting lattice parameter, a desired number of calculation steps, and a percent variation max and min of the lattice parameter. I think its really useful to comment code and inline calculations verbosely to reduce wasted time and headaches when referring to code in the future; plus it's kinda fun if you as me!

```
#                         a0_min         a0          a0_max
#                           |<- N_steps ->|<- N_steps ->|
```

12.3 Text decorations for utility and fun

Text decorations may sounds silly at the outset, and some of course can be. A good example of silly (but still super useful for putting a personalized touch on your own programs and scripts) text decorations might be the site kammerl.de/ascii/AsciiSignature.php. The tools there let you make and customize all sorts of ASCII art banners! An example of this is the following just saying 'Script' in the font called Sub-Zero from the site I just mentioned:

```
 ______     ______     ______     __     ______   ______
/\  ___\   /\  ___\   /\  == \   /\ \   /\  == \ /\__  _\
\ \___  \  \ \ \____  \ \  __<   \ \ \  \ \  _-/ \/_/\ \/
 \/\_____\  \ \_____\  \ \_\ \_\  \ \_\  \ \_\      \ \_\
  \/_____/   \/_____/   \/_/ /_/   \/_/   \/_/       \/_/
```

A slightly less silly use of text decorations is in making easily visible separations between different sections of a program or script. Some users may be familiar with the new section tag %% in MATLAB but BASH doesn't have such a nice functionality. One way of getting around this to at least the same visual effect is to put decorative and highly visible sections of commented out text or symbols into your code that separate blocks with independent functions so scrolling through or navigating the script at a glance is simpler. I include 'decorations' like these in most of my scripts because I find it aids in my own readability of the script, especially when I'm returning to the script after a number of months or years and I don't have a perfect memory of what I was intending to do at the time. Some examples are included below:

The following you'll have no doubt seen in many of the examples enclosed in his text, there's something I can't quite put my finger on about the pattern of alternating colons and dashes that I find very aesthetically pleasing to the eye.

```
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:- Block of Code -:-:-:-:-:-:-:-:-:-:-:-:-
#-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-
```

```
#====================================================================
#============================== Block of Code: ======================
#====================================================================
```

The following I especially like for LaTeX(note the % comments)

when I have whole chapters in individual .tex documents, large comments like these lend visual aid to break up the monotony on the screen and be able to more easily see where new sections begin or end.

```
% <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
%     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
% <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
%     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
% <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
%     <!>     <!>     <!>     <!>       NEW SECTION   <!>     <!>     <!>     <!>     <!>     <!>
% <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
%     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
% <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
%     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
% <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
```

While obviously not strictly necessary, I find these to be useful and also fun because you can personalize them to whatever suits your tastes! You can personalize these to a massive extent and add a bit of visual appeal and style that makes your code instantly recognizable and all your own.

```
%     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
%  S       E       B      <!>      S       F       S       C      <!>     <!>     <!>     <!>
%      T       D       O      <!>      H       O       C       O      <!>     <!>     <!>     <!>
% <!>      E       W       P      <!>      E       R       I       M      <!>     <!>     <!>
%     <!>      V       A       P      <!>      L      <!>      E       P      <!>     <!>     <!>
% <!>     <!>      E       R      <!>     <!>      L      <!>      N       U      <!>     <!>
%     (c)     <!>      N       D      <!>     <!>     <!>     <!>      T       T      <!>     <!>
% <!>      2      <!>     <!>     <!>     <!>     <!>     <!>     <!>      I       I      <!>
%     <!>      0      <!>     <!>     <!>     <!>     <!>     <!>     <!>      F       N      <!>
% <!>     <!>      2      <!>     <!>     <!>     <!>     <!>     <!>     <!>      I       G
%     <!>     <!>      2      <!>     <!>     <!>     <!>     <!>     <!>     <!>      C      <!>
% <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>     <!>
```

I especially love the almost wallpaper-like appearance of this design of my own!

12.4 Using text to speech and alert sounds in the terminal

One interesting bit of functionality that you can build into your scripts is to include audio output (assuming that the speaker kernel is loaded and working properly). One of the possibilities for this which can be used for all sorts of notifications is the OS X text to speech function named 'say'. For example, if you wanted the text to speech program to say aloud the word 'beep' then you could enter the following command into the OS X terminal:

```
say beep
```

The say command then allows for all sorts of useful monologue from your terminal session because you can store output from various parts of a script as variables and them pass them to the say command. I find this to be useful when I want to know if a script has completed or not without having to check on it periodically. However, I do actually find the text to speech more intrusive and startling than the default terminal chime. That chime is in fact what I use the most frequently when I desire this sort of functionality in my own scripts.

There are several ways to produce a small chime in the terminal. One is to echo the special character \007 (known as ASCII BEL \007) to the terminal with the following command:

```
echo -ne '\007'
```

Another way to achieve the same effect is to give bel as an argument to the tput command as follows:

```
tput bel
```

My preferred method is with the ASCII BEL command because it seems to work for me no matter what system I try it on (as long as the speaker kernel is loaded and working). The bel character is one of ASCII control characters which constitute ASCII characters 0-31. The bel character used to actually ring a physical bell on some systems. Coverage of all these control characters is beyond the scope of this text but they are of great interest. In fact, we have already spoken about one of these characters before in this text! In section 3.13 we talked about the carriage return which is actually another control character ASCII \013 and how it would cause trouble for us when we were trying to issue certain UNIX commands to a DOS-formatted file. Many of the original control characters have fallen out of common use however. /par

13 Concluding Remarks

The scope of this text has been to review some general use cases of the BASH shell and how it can be used for scientific computing and automation. The provided examples are intended to span a wide range of applications with tunability and modularity in mind. The tools provided in the previous sections are intended to be a launchpad for the reader's quick entry into the field of shell scripting and how it can make their life easier and their work more expansive. My hope for this text is that it can reduce the barriers to entry for scientists, engineers, and enthusiasts who want to begin investigations using scientific computing and make learning this skill fun and easy with practical examples.

This text is supplied free of charge and all are welcome to join. It's been my delight to share with you.

References

[1] Steven Edward Bopp, Haoliang Qian, and Zhaowei Liu. Influence of Hafnium Defects on the Optical and Structural Properties of Zirconium Nitride. *Physica Status Solidi (RRL) – Rapid Research Letters*, 2100372:1–8, 2021.

[2] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L. Chiarotti, Matteo Cococcioni, Ismaila Dabo, Andrea Dal Corso, Stefano De Gironcoli, Stefano Fabris, Guido Fratesi, Ralph Gebauer, Uwe Gerstmann, Christos Gougoussis, Anton Kokalj, Michele Lazzeri, Layla Martin-Samos, Nicola Marzari, Francesco Mauri, Riccardo Mazzarello, Stefano Paolini, Alfredo Pasquarello, Lorenzo Paulatto, Carlo Sbraccia, Sandro Scandolo, Gabriele Sclauzero, Ari P. Seitsonen, Alexander Smogunov, Paolo Umari, and Renata M. Wentzcovitch. QUANTUM ESPRESSO: A modular and open-source software project for quantum simulations of materials. *Journal of Physics Condensed Matter*, 21(39), 2009.

[3] P Giannozzi, Oliviero Andreussi, T Brumme, O Bunau, M Buongiorno Nardelli, M Calandra, R Car, C Cavazzoni, D Ceresoli, M Cococcioni, and others. Advanced capabilities for materials modelling with Quantum ESPRESSO. *Journal of Physics: Condensed Matter*, 29(46):465901, 2017.

[4] Torbjörn Björkman. CIF2Cell: Generating geometries for electronic structure programs. *Computer Physics Communications*, 182(5):1183–1186, 2011.

[5] Anton Kokalj. Computer graphics and graphical user interfaces as tools in simulations of matter at the atomic scale. *Computational Materials Science*, 28(2):155–168, 2003.

[6] Christian Vorwerk, Caterina Cocchi, and Claudia Draxl. Addressing electron-hole correlation in core excitations of solids: An all-electron many-body approach from first principles. *Physical Review B*, 95(15):1–10, 2017.

[7] Christian Vorwerk, Benjamin Aurich, Caterina Cocchi, and Claudia Draxl. Bethe-Salpeter equation for absorption and scattering spectroscopy: Implementation in the exciting code. *Electronic Structure*, 1(3), 2019.

[8] Dmitrii Nabok, Andris Gulans, and Claudia Draxl. Accurate all-electron G0W0 quasiparticle energies employing the full-potential augmented plane-wave method. *Physical Review B*, 94(3):1–9, 2016.

[9] Stephan Sagmeister and Claudia Ambrosch-Draxl. Time-dependent density-functional theory. *Physical Chemistry Chemical Physics*, 11(22):4436, 2009.

[10] Ronaldo Rodrigues Pela, Ute Werner, Dmitrii Nabok, and Claudia Draxl. Probing the LDA-1/2 method as a starting point for G0W0 calculations. *Physical Review B*, 235141(23):1–9, 2016.

[11] Pierre Hirel. Atomsk: A tool for manipulating and converting atomic data files. *Computer Physics Communications*, 197:212–219, 2015.

[12] G. Kresse and J. Furthmüller. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Physical Review B*, 54(16), 1996.

[13] G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, 6(1):15–50, 1996.

[14] G. Kresse and J. Hafner. Ab initio molecular dynamics for liquid metals. *Physical Review B*, 47(1):558–561, 1993.

[15] Koichi Momma and Fujio Izumi. VESTA : a three-dimensional visualization system for electronic and structural analysis . *Journal of Applied Crystallography*, 41(3):653–658, 5 2008.

[16] Andrew Ian Duff, M. W. Finnis, Philippe Maugis, Barend J. Thijsse, and Marcel H.F. Sluiter. MEAMfit: A reference-free modified embedded atom method (RF-MEAM) energy and force-fitting code. *Computer Physics Communications*, 196:439–445, 2015.

[17] P. Brommer and F. Gähler. Effective potentials for quasicrystals from ab-initio data. *Philosophical Magazine*, 86(6-8):753–758, 2006.

[18] Peter Brommer and Franz Gähler. Potfit: Effective potentials from ab initio data. *Modelling and Simulation in Materials Science and Engineering*, 15(3):295–304, 2007.

[19] Peter Brommer, Alexander Kiselev, Daniel Schopf, Philipp Beck, Johannes Roth, and Hans Rainer Trebin. Classical interaction potentials for diverse materials from ab initio data: A review of potfit. *Modelling and Simulation in Materials Science and Engineering*, 23(7), 2015.

[20] Daniel Schopf, Peter Brommer, Benjamin Frigan, and Hans Rainer Trebin. Embedded atom method potentials for Al-Pd-Mn phases. *Physical Review B - Condensed Matter and Materials Physics*, 85(5):1–8, 2012.

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>   <!>   <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
```

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>   <!>   <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
```

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>   <!>   <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
```

```
   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>
<!>    S     F     S     C    <!>    S     E     B    <!>   1st   <!>   <!>   <!>
   <!>    H     O     C     O    <!>    T     D     O    <!>    E    <!>   <!>
<!>   <!>    E     R     I     M    <!>    E     W     P    <!>    d    <!>   <!>
   <!>   <!>    L    <!>    E     P    <!>    V     A     P    <!>    n    <!>
<!>   (c)   <!>    L    <!>    N     U    <!>    E     R    <!>   <!>   <!>   <!>
   <!>    2    <!>   <!>   <!>    T     T    <!>    N     D    <!>   <!>   <!>
<!>   <!>    0    <!>   <!>   <!>    I     I    <!>   <!>   <!>   <!>   <!>   <!>
   <!>   <!>    2    <!>   <!>   <!>    F     N    <!>   <!>   <!>   <!>   <!>
<!>   <!>   <!>    2    <!>   <!>   <!>    I     G    <!>   <!>   <!>    E    <!>
   <!>   <!>   <!>   <!>   <!>   <!>   <!>    C    <!>   <!>   <!>   <!>    O
<!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>   <!>    F
```