PHY 392Q

Density Functional Theory

Feliciano Giustino

Department of Physics & Oden Institute For Computational Engineering and Sciences
The University of Texas at Austin

Hands-On Tutorials based on Quantum ESPRESSO and the Lonestar6 supercomputer at the Texas Advanced Computing Center (TACC). We gratefully acknowledge the Quantum ESPRESSO community for their continuous commitment to open, collaborative, and high-quality scientific software.

The University of Texas at Austin, Fall 2025

Hands-On 0
Connecting to TACC

Multi-factor authentication

In this class we will use the Lonestar6 (LS6) supercomputer at TACC. The reason for this choice is that by using LS6 we will all work within the same environment, which makes it easier to compare results, identify and resolve potential problems.

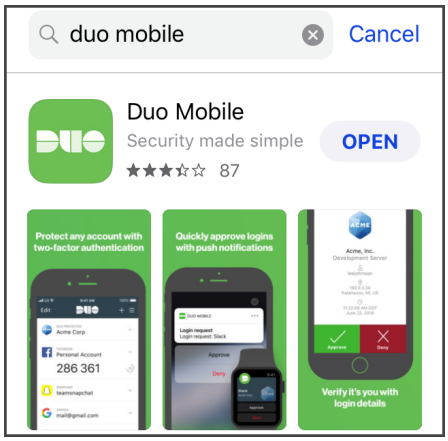
In order to access TACC resources you will need (i) a TACC account, and (ii) to set up multi-factor authentication (MFA).

If you do not have a TACC account, please register at the page:

https://accounts.tacc.utexas.edu/register

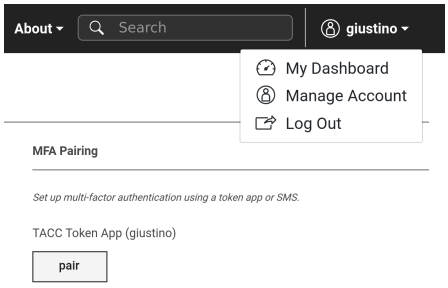
Once you are registered, please let me know so I can enable you on the allocation that has been assigned to our class. For this I will need your TACC user ID.

To set up MFA we will use DUO, which is the same app used for other UT resources. Download and install the DUO app on your phone if you don't already have one:



Then go to https://tacc.utexas.edu/portal/account

and click on your name/Manage Account. You will see the following:



Click on "pair" and scan the QR code. Once you are done, your DUO app will show a 6-digits code that you will use every time you log in to LS6.

Linux and macOS

If you have a laptop or desktop computer running on Linux (e.g. Ubuntu) or macOS, then you need to search for the `Terminal` app in the Dock. Once you open the app you will see something like (for Ubuntu Linux)

We check that we can see the computers at TACC:

```
$ ping ls6.tacc.utexas.edu
PING ls6.tacc.utexas.edu (129.114.62.201) 56(84) bytes of data.
64 bytes from login1.ls6.tacc.utexas.edu (129.114.62.201): icmp_seq=1 ttl=53 time=3.40 ms
```

If you can see the ping response then you are all set and we can proceed to HandsOn 1.

Windows

If you are using Windows on your laptop/desktop, then in order to connect to TACC you will need a software that can handle a secure shell (SSH) connection.

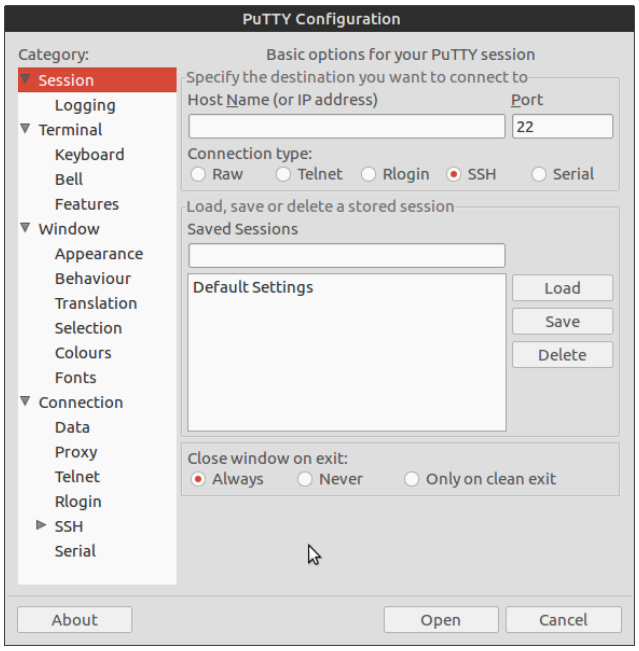
A popular choice is `Putty`, which can be downloaded from:

`https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html`

You most likely need the Putty version for 64 bit architecture. However, if you are unsure whether you have a 32 bit or 64 bit architecture, then you can check by clicking on the menus:

Control Panel > System and Security > System

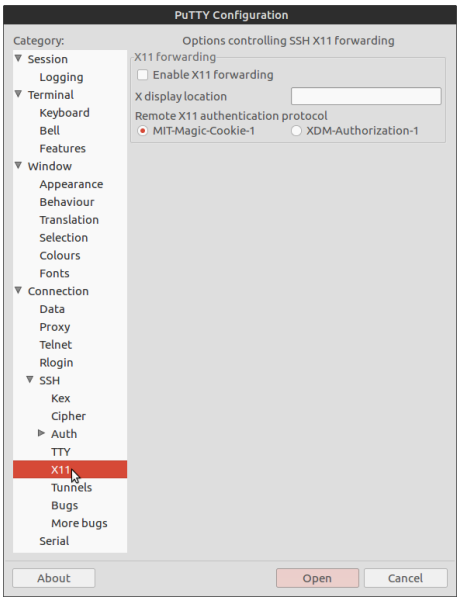
Upon executing `Putty`, you will see something like the following:

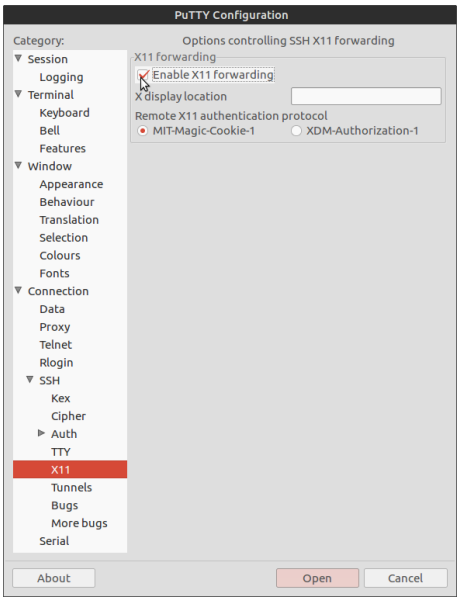


In the field 'Host Name' we enter:

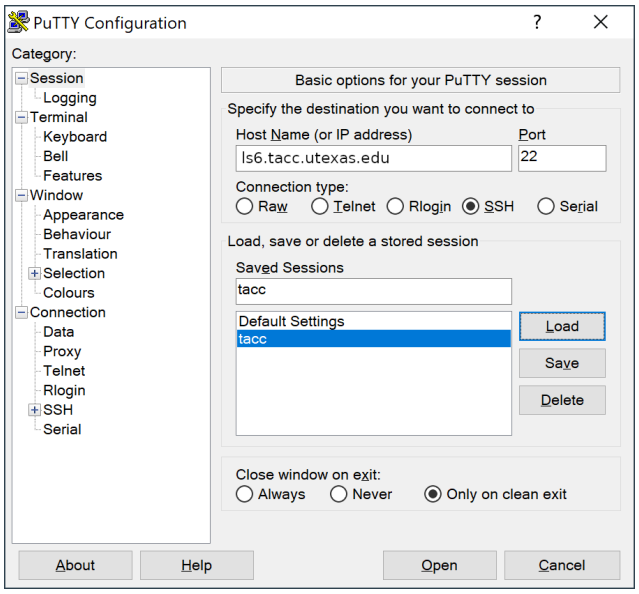
`ls6.tacc.utexas.edu`

In order to be able to see graphics over this connection, we need to enable 'X11 forwarding'. For this we proceed as indicated below:





Now we can save these settings, so that next time we will just click on the session name, say 'tacc':



Visualizing graphics from a Windows machine

In order to visualize graphics when using Putty, your machine must be able to understand the X11 protocol. This can be done by downloading the program Xming.

The installation file can be found at the following link:

`https://sourceforge.net/projects/xming/files/Xming/6.9.0.31/Xming-6-9-0-31-setup.exe/download`

After installing `Xming` the procedure for running calculations and visualizing graphics at TACC is as follows:

- Start `Xming`. This application will now run in the background.
- Start `Putty` and open a session.

From this point onward everything works exactly in the same way as for users of Linux or macOS.

Visualizing graphics from a Mac

As for Windows, to visualize graphics when you are connected to TACC, you will need to install an extra bit of software called `Xquartz`:

`https://www.xquartz.org`

Also in this case, after installation:

- Start `Xquartz`. This application will now run in the background.
- Open a Terminal.

Now you can continue as for Linux users.

Hands-On 1
Setting up LS6

Login shell and compilation

We will perform calculations on the Lonestar6 (LS6) system of TACC. LS6 consists of 560 CPU nodes and 88 GPU nodes. The CPU nodes have two 64-core AMD EPYC Milan with 256 GB of RAM. This means 128 physical cores with 2GB/core. The GPU nodes also have the same AMD Milan host device and 3 NVIDIA A100 accelerators each. The peak performance of this machine is 5 TFlops.

During this class we will use CPU nodes. If time permits, we might also try experimenting with GPU nodes later.

Modern DFT-based calculations can require anything from a hundred cores to hundreds of thousands of cores, but in this class we will use between 4 and 24 cores at a time.

In order to work on LS6 we need to establish a secure connection. We first open a terminal (on Ubuntu/Mac machines; from Windows we launch Putty), then we type:

```
$ ssh -X ls6.tacc.utexas.edu -l giustino
```

where giustino must be replaced by the username that you have been assigned. After entering your password and the verification code received on your mobile, you will see something like:

```
Last login: Mon Sep  1 17:31:31 2025 from 128.62.160.66
--------------------------------------------------------------------------------
Welcome to the Lonestar6 Supercomputer
Texas Advanced Computing Center, The University of Texas at Austin
--------------------------------------------------------------------------------
...

Welcome to Lonestar6, please read these important system notes:

06/18/2025: /work access has been restored.

--> Lonestar6 user documentation is available at:
https://portal.tacc.utexas.edu/user-guides/lonestar6

---------------------- Project balances for user giustino ----------------------
| Name           Avail SUs     Expires | Name           Avail SUs     Expires |
| DMR21056           90370  2026-04-30 | DMR23030            5000  2026-08-31 |
------------------------ Disk quotas for user giustino -------------------------
| Disk         Usage (GB)     Limit    %Used   File Usage       Limit   %Used |
| /scratch            0.0       0.0     0.00            0           0    0.00 |
| /home1              1.7      10.0    17.30        11364           0    0.00 |
| /work               2.1    1024.0     0.21        20021     3000000    0.67 |
--------------------------------------------------------------------------------
...
```

```
login1.ls6(1)$
```

We can customize the Unix 'shell' environment by shortening the prompt, creating a couple of 'aliases', and adding modules that we will need later on. We copy/paste the following into the terminal (it is important to copy/paste exactly as it is, since the `bash` shell is very strict with spaces):

```
cat >> .bashrc << EOF
PS1="$ "
alias c="clear"
module load intel/19.1.1
module load impi/19.0.9
calc()  awk "BEGIN print $* ";
EOF
source ~/.bashrc
```

From now on the prompt will be simply '`$`' and the command '`c`' and '`l`' will clear the screen and list the content of a directory, respectively.

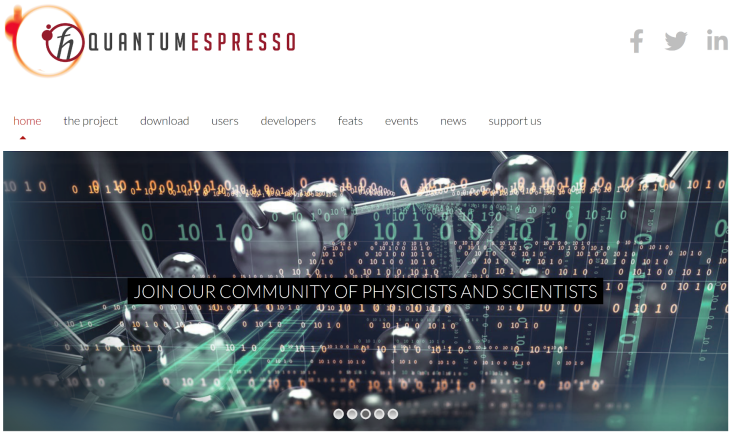
The `module load` directives make sure that you have the correct programming environment to compile Quantum ESPRESSO. In case you want to check the modules loaded in your environment, just type `module list`.

We can now create a folder for this school (`mkdir`) in our home directory and move inside (`cd`):

```
$ mkdir ~/PHY392Q ; cd ~/PHY392Q
```

In this school we will be using the `Quantum ESPRESSO` (QE) software package. QE is an open-source suite of *ab initio* electronic structure codes based on pseudopotentials and planewaves.

The project website can be found at `www.quantum-espresso.org`



Note QE is already installed on LS6 as precompiled executables. The reason for downloading and compiling this program by ourselves is to learn how to get started in case QE was not already available. This will allow you to work on your own computer or other systems after the user accounts on TACC will have been deactivated.

To use QE we download the latest release as a compressed archive, and we install it in our working directory. The latest snapshot can be found at http://www.quantum-espresso.org (requires free registration), or on GitLab. In this class we will use the latest release QE 7.5 (August 2025):

```
$ wget https://gitlab.com/QEF/q-e/-/archive/qe-7.5/q-e-qe-7.5.tar.gz
```

Then we unzip and unpack the compressed archive (`tar xfz`):

```
$ tar xfz q-e-qe-7.5.tar.gz
```

QE is now unpacked. It is useful to take a look inside the directory:

```
$ cd q-e-qe-7.5 ; ls -lh
```

```
total 208K
...
drwx------   2 giustino G-821787   53 Aug 19 04:57 archive
drwx------   6 giustino G-821787  168 Aug 19 04:57 atomic
drwx------   3 giustino G-821787 4.0K Aug 19 04:57 cmake
-rw-------   1 giustino G-821787  30K Aug 19 04:57 CMakeLists.txt
-rwx------   1 giustino G-821787 2.6K Aug 19 04:57 configure
-rw-------   1 giustino G-821787 1.1K Aug 19 04:57 CONTRIBUTING.md
drwx------   6 giustino G-821787  145 Aug 19 04:57 COUPLE
drwx------   6 giustino G-821787  141 Aug 19 04:57 CPV
drwx------   4 giustino G-821787 4.0K Aug 19 04:57 dev-tools
drwx------   2 giustino G-821787  225 Aug 19 04:57 dft-d3
drwx------   4 giustino G-821787 4.0K Aug 19 04:57 Doc
drwx------   8 giustino G-821787  233 Aug 19 04:57 EPW
drwx------   8 giustino G-821787  171 Aug 19 04:57 PHonon
drwx------   4 giustino G-821787   91 Aug 19 04:57 PIOUD
drwx------   7 giustino G-821787  151 Aug 19 04:57 PP
drwx------   2 giustino G-821787 4.0K Aug 19 04:57 pseudo
drwx------   7 giustino G-821787  158 Aug 19 04:57 PW
drwx------ 115 giustino G-821787 8.0K Aug 19 04:57 test-suite
...
```

In this class we are mainly interested in the program `pw.x`, which is contained in the folder `PW`. In order to use this program we need to compile the Fortran source into an executable. This operation is performed by the script `Makefile`. `Makefile` in turn needs to know where to look for compilers and numerical libraries. This information is retrieved by the program `configure`, which explores the host system for available software. To make this happen we issue:

```
$ ./configure ; make pw
```

This operation requires approximately 6 min.

While we wait we might as well check the page of TACC where LS6 is described: portal.tacc.utexas.edu/user-guides/lonestar6.

At the end of the compilation we should find a pointer to the newly-created executable `pw.x` inside the directory `bin`:

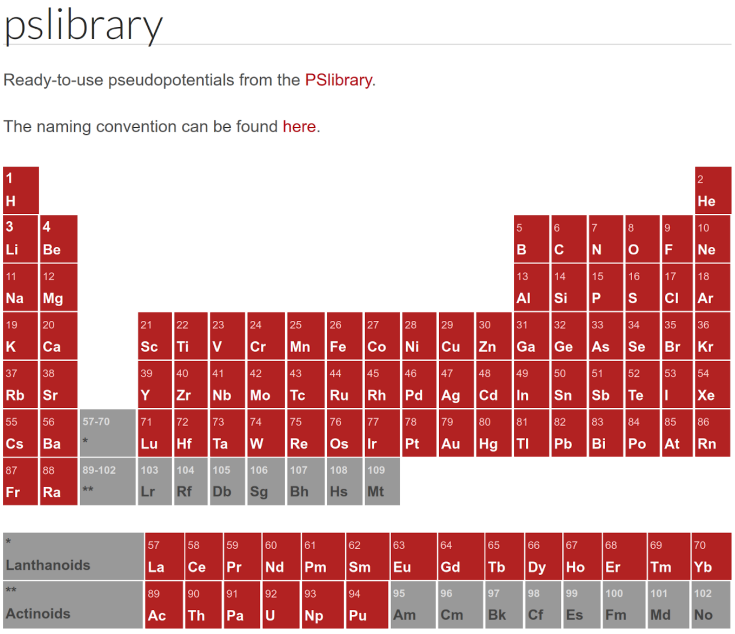
```
$ ls -lth bin
```

```
lrwxrwxrwx 1 giustino G-821787 14 Sep  1 19:43 pw.x -> ../PW/src/pw.x
```

Test run

Now we want to execute a simple job on the cluster. The goal of this operation is to make sure that everything runs smoothly.

We will consider a simple total energy calculation for a **silicon** crystal in the diamond structure. In order to proceed we first need a *pseudopotential*. The concept of pseudopotentials is covered in the theory lectures; for now it suffices to know that we need one pseudopotential for each atomic species, and that the pseudopotential describes the atomic nucleus and all the electrons except the outermost (valence) shell. The QE pseudopotential libraries can be found at http://www.quantum-espresso.org/pseudopotentials. There are many varieties of pseudopotentials, for example this is how the clickable PS library looks like:



Here we focuse on the original pseudopotentials developed for the QE code at http://pseudopotentials.quantum-espresso.org/legacy_tables/original-qe-pp-library. This choice is not optimal for production runs, but we stick to it because these are the pseudos that I used to generate my reference results.

If we click on silicon we see a list of available pseudopotentials. In this example we will use the pseudopotential labelled Si.pz-vbc.UPF. By hovering on this link with the mouse we can copy/paste the web link, and we can use it to download the file directly in our home directory:

```
$ cd ~/PHY392Q
$ wget http://pseudopotentials.quantum-espresso.org/upf_files/Si.pz-vbc.UPF
```

Now we should have the pseudopotential file. We can look inside this plain text file by using the command `more`:

```
$ more Si.pz-vbc.UPF

<UPF version="2.0.1">
<PP_INFO>
Generated by new atomic code, or converted to UPF format
```

```
Author:
Generation date:
Pseudopotential type: NC
Element: Si
Functional:  SLA  PZ   NOGX NOGC
Suggested minimum cutoff for wavefunctions:   0. Ry
Suggested minimum cutoff for charge density:   0. Ry
The Pseudo was generated with a Non-Relativistic Calculation
L component and cutoff radius for Local Potential:  0   0.0000
Valence configuration:
nl pn  l   occ       Rcut    Rcut US       E pseu
3S  0  0  2.00      0.000      0.000     0.000000
3P  0  1  2.00      0.000      0.000     0.000000
Generation configuration: not available.
</PP_INFO>
<!--
END OF HUMAN READABLE SECTION
-->
...
```

Note For all production calculations it is recommended to use pseudopotentials from the 'PSlibrary table' or from the 'Standard Solid State PPs' which contain up-to-date pseudopotentials:

http://www.quantum-espresso.org/pseudopotentials

An additional library that is very popular is the PseudoDojo library:

http://www.pseudo-dojo.org

It is a universal rule of supercomputing centers that **users should never perform calculations in their home directory**. In order to execute `pw.x` we move to a scratch folder:

```
$ cd $SCRATCH
```

We now copy (`cp`) the pseudopotential file in the scratch folder, and for simplicity we also copy the executable in the same folder (this is not standard practice but it makes things easier to understand at the beginning):

```
$ cp ~/PHY392Q/Si.pz-vbc.UPF ./
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x ./
```

At this point we have the executable `pw.x` and the pseudopotential for silicon `Si.pz-vbc.UPF`. We are missing the input file for the executable.

We can create the simplest possible input file, `silicon-1.in`, as follows:

```
$ cat << EOF > silicon-1.in
&control
 calculation = "scf",
 prefix = "silicon",
```

```
 pseudo_dir = "./",
 outdir = "./"
/
&system
 ibrav = 2,
 celldm(1) = 10.28,
 nat = 2,
 ntyp = 1,
 ecutwfc = 18.0,
/
&electrons
/
ATOMIC_SPECIES
 Si  28.086  Si.pz-vbc.UPF
ATOMIC_POSITIONS alat
 Si 0.00 0.00 0.00
 Si 0.25 0.25 0.25
K_POINTS automatic
 4 4 4 1 1 1
EOF
```

For future reference, it is convenient to keep a copy of this file in our `$HOME` folder:

```
$ mkdir ~/PHY392Q/HandsOn01
$ cp silicon-1.in ~/PHY392Q/HandsOn01/
```

Keeping some important files in the home directory is useful because the scratch space is only meant for temporary files and is periodically cleared by TACC.

In order to run our jobs we will mostly use interactive sessions on LS6. An interactive session is launched by issuing the following command:

```
$ idev -m 90 -A DMR23030
```

Where the flag `-m 90` specify that we want to use this session for 90 minutes (the default is 30m). This session will give us access to one node, i.e. 124 cores. The flag `-A DMR23030` specifies the account that we will be charged at TACC (for this course I requested an 'educational' account named 'PHY392Q' which has been assigned a charging code 'DMR23030'). If this is the only allocation that you have at TACC, then the flag `-A DMR23030` is not necessary.

Important Note: We run the `idev` command **only once** at the beginning of the session.

When the interactive session starts, we can run `pw.x` by issuing:

@@@

```
$ ibrun -n 4 pw.x < silicon-1.in > silicon-1.out
```

The command `ibrun -n 4 pw.x` specifies that we want to run `pw.x` on 4 cores. We can also say that we initiate 4 MPI processes, where MPI is the Message Passing Interface invoked by `ibrun` (which in turn is a wrapper for `mpirun` and `mpiexec` depending on the compilers used). We 'feed' the input file using the input redirection command `<`. We

place the output of the entire command `ibrun -n 4 pw.x < silicon-1.in` into the file `silicon-1.out` using the output redirect `>`.

This job will take less than a second to complete. The output file is `silicon-1.out` and should look like the following:

```
$ more silicon-1.out
```

```
TACC:  Starting up job 2610784
TACC:  Starting parallel tasks...

     Program PWSCF v.7.5 starts on  7Sep2025 at 18:30:33

     This program is part of the open-source Quantum ESPRESSO suite
     for quantum simulation of materials; please cite
         "P. Giannozzi et al., J. Phys.:Condens. Matter 21 395502 (2009);
         "P. Giannozzi et al., J. Phys.:Condens. Matter 29 465901 (2017);
         "P. Giannozzi et al., J. Chem. Phys. 152 154105 (2020);
          URL http://www.quantum-espresso.org",
     in publications or presentations arising from this work. More details at
     http://www.quantum-espresso.org/quote

     Parallel version (MPI), running on     4 processors

     MPI processes distributed on     1 nodes
     224130 MiB available memory on the printing compute node when the environment starts

     Waiting for input...
     Reading input from standard input
...
   PWSCF        :      0.08s CPU      1.66s WALL

   This run was terminated on:  18:30:34   7Sep2025
=------------------------------------------------------------------------------=
   JOB DONE.
=------------------------------------------------------------------------------=
TACC:  Shutdown complete. Exiting.
```

Note Interactive sessions are mostly used for software development purposes. For most production jobs we do not run interactive sessions, instead we use a *batch queuing system*. In this case we prepare a 'job' submission script that instructs the machine about the resources that we need, and then we place it in a queue with all the other jobs pending. A job scheduler (SLURM in LS6) monitors the available resources on the cluster and allocates them in order of priority.

This route is not worth the effort for small, fast jobs, but it becomes a necessity for jobs that are expected to require significant resources such as many nodes over several hours.

If we want to follow this route we need to create a submission script `job-1.pbs` as follows:

```
$ cat << EOF > job-1.pbs
#!/bin/bash
#SBATCH -J myjob
#SBATCH -o myjob.o%j
#SBATCH -e myjob.e%j
#SBATCH -N 1
#SBATCH -n 4
#SBATCH -p development
#SBATCH -t 00:10:00
#SBATCH -A DMR23030

ibrun -n 4 pw.x < silicon-1.in > silicon-1.out
EOF
```

The `SBATCH` directives are used to request the appropriate resources, as we did for the interactive session. `-J` is for the job name, `-o` is the output file, `-e` is the error file (reporting on possible issues with the eecution), `-N` the number of nodes, `-n` the number of MPI tasks, `-p` the queue (either 'normal' or 'development'), `-t` the reservation time (hh:mm:ss).

We submit this job to the queue by issuing:

```
$ sbatch job-1.pbs
```

We can check the status of this job in the queue using the command `squeue`. If we do not remember our username we can find this information using:

```
$ whoami
giustino
$ squeue -u giustino
             JOBID       PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           3002024     development    myjob giustino  R       0:04      1 nid00009
```

At the end of the execution we will find the results in the file `silicon-1.out`, as for the interactive session.

Note: For small jobs up to 16 cores, it is preferabe to use the queue `vm-small` which reserves only 16 cores out of 128 of a given node (and charges you accordingly).

Generating new input files

In the following we will prepare input files by modifying the file `silicon-1.in`.

Instead of using the `cat` command as we did in the previous section, we first create a new file by simply copying the previous file:

```
$ cp silicon-1.in silicon-2.in
```

Now we use `vi` to modify the as-created file:

```
$ vi silicon-2.in
```

This command will open the file inside the current terminal window. The rules for using `vi` are simple:

1 We move around using [↑] [↓] [→] [←]
2 In order to change the text we press [i] (insert) and modify as needed
3 When we are done making changes we press [ESC]
4 We write the modified file and exit by typing [:] [w] [q] (`w` is for write, `q` is for quit)

As an example we now change the parameter `ecutwfc` from 18 Ry to 40 Ry. This parameter represents the kinetic energy cutoff of the planewaves used in the Fourier expansion of the electron wavefunctions (see theory lectures). We can also change the Brillouin zone sampling from `4 4 4 1 1 1` to something more accurate, say `8 8 8 1 1 1`.

We execute once again `pw.x` using the new input file:

```
$ ibrun -n 4 pw.x < silicon-2.in > silicon-2.out
```

The calculation parameters and runtime flags will become more clear as we progress with the hands-on sessions.

Documentation

A comprehensive description of the input variables accepted by `pw.x` can be found here:

https://www.quantum-espresso.org/Doc/INPUT_PW.html


Input File Description
Program: pw.x / PWscf / Quantum Espresso

TABLE OF CONTENTS

INTRODUCTION

&CONTROL

calculation | title | verbosity | restart_mode | wf_collect | nstep | iprint | tstress | tprnfor | dt | outdir | wfcdir | prefix | lkpoint_dir | max_seconds | etot_conv_thr | forc_conv_thr | disk_io | pseudo_dir | tefield | dipfield | lelfield | nberrycyc | lorbm | lberry | gdir | nppstr | lfcpopt | gate

&SYSTEM

ibrav | celldm | A | B | C | cosAB | cosAC | cosBC | nat | ntyp | nbnd | tot_charge | starting_charge | tot_magnetization | starting_magnetization | ecutwfc | ecutrho | ecutfock | nr1 | nr2 | nr3 | nr1s | nr2s | nr3s | nosym | nosym_evc | noinv | no_t_rev | force_symmorphic | use_all_frac | occupations | one_atom_occupations | starting_spin_angle | degauss | smearing | nspin | noncolin | ecfixed | qcutz | q2sigma | input_dft | ace | exx_fraction | screening_parameter | exxdiv_treatment | x_gamma_extrapolation | ecutvcut | nqx1 | nqx2 | nqx3 | localization_thr | lda_plus_u | lda_plus_u_kind | Hubbard_U | Hubbard_J0 | Hubbard_V | Hubbard_alpha | Hubbard_beta | Hubbard_J | starting_ns_eigenvalue | U_projection_type | Hubbard_parameters | ensemble_energies | edir | emaxpos | eopreg | eamp | angle1 | angle2 | lforcet | constrained_magnetization | fixed_magnetization | lambda | report | lspinorb | assume_isolated | esm_bc | esm_w | esm_efield | esm_nfit | fcp_mu | vdw_corr | london | london_s6 | london_c6 | london_rvdw | london_rcut | dftd3_version | dftd3_threebody | ts_vdw_econv_thr | ts_vdw_isolated | xdm | xdm_a1 | xdm_a2 | space_group | uniqueb | origin_choice | rhombohedral | zgate | relaxz | block | block_1 | block_2 | block_height

&ELECTRONS

A comprehensive description of the QE project and the most recent developments is provided in the following manuscript:

P. Giannozzi et al., Advanced capabilities for materials modelling with Quantum ESPRESSO, J. Phys.: Condens. Matter 29, 465901 (2017).

Information about the job scheduling system of LS6 can be found at:

https://docs.tacc.utexas.edu/hpc/lonestar6/#running

In order to find out which 'queues' or 'partitions' are available we can use the command

```
$ sinfo -s
```

Unix cheat sheet

Below is a summary of useful Unix commands that we will use throughout this class:

`ls`	List content of current folder
`clear`	Clear the screen
`pwd`	Print working directory
`cd thisfolder`	Enter the folder called *thisfolder*
`cd ..`	Go up one folder
`cd`	Go to home directory
`more thisfile`	Show the content of the file called *thisfile*
`rm thisfile`	Remove the file called *thisfile*
`cp file1 file2`	Copy *file1* into *file2*
`mv thisfile thisfolder/`	Move the file *thisfile* into the folder *thisfolder*
`vi thisfile`	Open the file *thisfile* using the Vi text editor

`grep thisword thisfile`	Search for the word *thisword* inside the file *thisfile*
`man thiscommand`	Show the manual page for the command *thiscommand*
`scp myfile username@remoteip:~/`	Copy the file *myfile* to the home directory of user *username* in the remote computer identified by the IP address *remoteip*.

To Do

To familiarize yourself with the basics, repeat all the the steps illustrated in this Hands On session, in particular:

1 Login into your account, set the modules in the file `.bashrc`, and create your working directory
2 Download the QE package, unzip, configure, and make the executable `pw.x`
3 Download the pseudopotential for silicon `Si.pz-vbc.UPF`
4 Create the input files `silicon-1.in`
5 Execute `pw.x` and check the output file `silicon-1.out`

It is strongly recommended that you type in (rather than copy/paste) all the instructions. If something goes wrong, you can directly copy/paste from this PDF document into the terminal (This is discouraged because by doing so you will not learn much).

Hands-On 2
Convergence and Scaling

Exercise 1

We want to familiarize ourselves with one important convergence parameter of DFT calculations based on pseudopotentials and planewaves, the **planewave kinetic energy cutoff** `ecutwfc`.

To this aim we request an interactive session, we go to the scratch space, and copy over the executable, pseudopotential, and input files generated in HandsOn 1:

```
$ idev -m 90  -A DMR23030
$ cd $SCRATCH
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x ./
$ cp ~/PHY392Q/Si.pz-vbc.UPF ./
$ cp ~/PHY392Q/HandsOn01/silicon-1.in ./silicon-3.in
```

Using `vi` we modify the input variable `ecutwf` to 5.0 Ry (note 1 Ry = 13.6058 eV). After this change, line #12 of `silicon-3.in` should read:

```
ecutwfc   = 5.0,
```

Now we execute `pw.x` as usual:

```
$ ibrun -n 4 pw.x < silicon-3.in > silicon-3.out
```

When the job is completed we can analyze the output file `silicon-3.out`. This output file contains the most important information regarding your run. During the hands-on exercise sessions we will gradually become familiar with the various sections of this file. For now we concentrate only on some basic information.

First of all we can check that we are using the local density approximation (LDA) to DFT. To see this, open the output file using `vi`, and search for the words `Exchange-correlation`. To activate the search function in `vi` we simply press `/` and enter the search word. You will find:

```
    Exchange-correlation      =  SLA  PZ   NOGX NOGC ( 1  1  0  0 0 0)
```

Here `SLA` stands for 'Slater exchange', `PZ` stands for Perdew-Zunger parametrization of the LDA, `NOGX` and `NOGC` say that density gradients are not taken into account (the meaning of this will become clear from the theory lectures). The numbers are internal codes of `pw.x`.

Now we search for the words `kinetic-energy cutoff`. This should be indentical to the value of `ecutwfc` that you specified in the input file. This parameter is the kinetic energy cutoff of the planewaves basis set, and sets the number of planewaves in which every Kohn-Sham wavefunction is expanded (i.e. the number of Fourier components used to represent electron wavefunctions).

The number of planewaves corresponding to the cutoff `ecutwfc` can be found by searching for the keyword `Parallelization info`. You will see the following line:

```
Parallelization info
--------------------
sticks:   dense  smooth     PW     G-vecs:    dense   smooth      PW
...
Sum          73      73     37                  411      411     137
```

This means that each wavefunction is expressed as a linear combination of 137 planewaves. To figure out what is the actual size of the **G**-vector grid, we can look for the keyword `FFT dimensions`:

```
Dense  grid:      411 G-vectors     FFT dimensions: (  12,  12,  12)
```

This means that the grid consists or $12\times12\times12$ vectors.

▶ Can you explain why the grid seems to contain $12\times12\times12 = 1728$ **G**-vectors, but above we have seen that the number of planewaves used for the wavefunctions is only 137?

The number of electrons and wavefunctions is found by searching for the keywords `number of electrons`, and `number of Kohn-Sham states`. We find:

```
number of electrons       =         8.00
number of Kohn-Sham states=            4
```

meaning that we have 8 electrons in 4 Kohn-Sham states. This is consistent with the fact that in silicon we have no spin-polarization, so each spatial wavefunction describes two electrons (one spin-up and one spin-down). From the Periodic Table we know that the silicon atom has 14 electrons. Here we only have the $3s$ and $3p$ electrons (4 in total) because the $1s$, $2s$, and $2p$ electrons (10 in total) are "fused" together with the nucleus inside the pseudopotential. We say that we have $3s$ and $3p$ **valence states**. The $1s$, $2s$, and $2p$ are called **core states**. The cores states are not described explicitly in this calculation: their effect is included indirectly via the pseudopotential file.

Now we want to look at the most important quantity in the output file, the DFT total energy. Search for the marker `!` in the output file. You should find:

```
!    total energy              =     -15.60437792 Ry
```

This value should be taken with caution: it contains an offset that depends on the chosen pseudopotentials and on some conventions in the code. This energy is **not** referred to vacuum (as for example when we solve the Schrödinger equation for the hydrogen atom), because there is no vacuum reference when we perform a calculation for an infinitely-extended crystal.

As a consequence, the absolute value of DFT total energy in extended solids is not meaningful; what is meaningful is **difference in total energy** between two configurations, because the the artificial offset cancels out when taking the difference.

Finally we look at the timing: search for the word '`WALL`'. You should find:

```
init_run     :      0.02s CPU      0.12s WALL (       1 calls)
...
PWSCF        :      0.06s CPU      4.37s WALL
```

The number on the left is the CPU-time, that is the execution time as measured on each individual CPU. The number on the right is the 'wall-clock' time, and indicates the actual time elapsed from the beginning to the end of the run. It includes the time spent in I/O operations. The code provides a breakdown of the time spent in each key subroutine, and the grand total at the end.

Now we want to study *how the total energy, the number of planewaves, and the execution time* vary as a function of the planewaves cutoff `ecutwfc`.

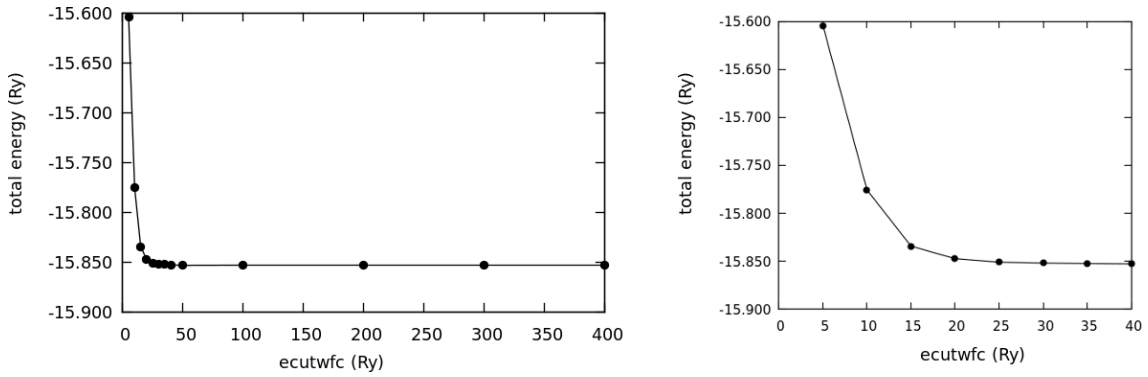
▶ Repeat the above steps by setting `ecutwfc` to 5, 10, 15, 20, 25, 30, 35, 40, 50, 100, 200, 300, 400 in the input file. It is convenient to generate separate input/output files and then search for the energy, the number of planewaves, and the CPU time in each output file. For this run we will use 4 CPUs, that is `ibrun -n 4 pw.x`.

For example, you can collect the results by creating a text file using `vi exercise1.txt` and entering your results one by one. You should be able to construct a file looking like this (omitted rows are for you to fill in):

```
$ more excercise1.txt

# ecutwfc (Ry)  planewaves   energy (Ry)   time (s)
    5               137     -15.60437789    0.06
   10               283     -15.77558457    0.07
  ...               ...              ...     ...
  400             40857     -15.85306926    3.59
```

At this point we can analyze our results. For this you can either use `gnuplot` directly on the cluster, or you can transfer the file `exercise1.txt` using the command `scp` or the program `filezilla`, and then plot the data using your favourite software (e.g. `Origin` or `Excel`).

For the total energy you should find something like this (left: full range; right: zoom)



Here we see that, using a cutoff of 25 Ry, we obtain a total energy which is only 15 meV/atom higher than our best-converged value at 400 Ry. This level of error is acceptable, and in practice 25 Ry is good enough for this exercise. Most quantities that can be computed using DFT **depend critically on this cutoff**, therefore we must always perform this test when running DFT calculations. In general, different properties (total energy, equilibrium structure, band structures, vibrations, etc.) will exhibit a slightly different convergence behavior, therefore it is very important to **always check** that a given property is converged with respect to the planewaves cutoff.

▶ Plot the data obtained and verify that the energy is well-converged at 25 Ry.

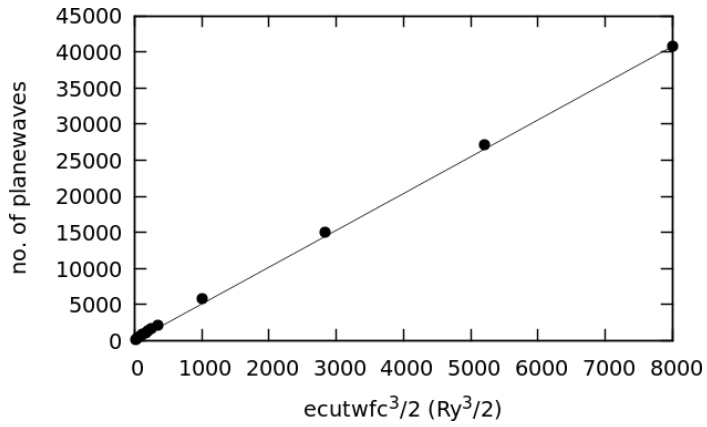
Since the planewaves cutoff is so important, can we not use a very large value to be on the safe side? The answer is that the higher the cutoff, the more time-consuming the calculation.

▶ To confirm the above point, plot the CPU time vs. the cutoff using the values inside the file `exercise1.txt`.

In this example the runtime is of the order of a few seconds, therefore we can use a very high cutoff without problems. However, in most DFT calculations a careful choice of cutoff can save us thousands of node-hours of compute time.

The longer times required for higher cutoffs relate to the fact that we are performing linear algebra operations using larger arrays to describe the electron wavefunctions.

► Verify that the number of planewaves increases with the cutoff. In particular, verify that a plot of the number of planewaves vs. `ecutwfc`$^{3/2}$ yields approximately a straight line:



► Can you explain the origin of this relation between the cutoff and the number of planewaves? (To answer this question you will need to look at Lecture 5).

Note: If you want to plot your data using `gnuplot`, here is the standard command line:

```
$ gnuplot
> plot "exercise1.txt" u 1:3 w lp pt 6 ps 2 lc 3
```

Here `u 1:3` indicates that we want to plot column 1 as x-coordinate and column 2 as y-coordinate. `w lp` means 'with line and points', `pt` and `ps` are the type and size of the points, respectively, and `lc 3` sets the line color to blue.

More information about `gnuplot` can be found at http://www.gnuplot.info/index.html

Note: If you do not want to change each input file manually, you can use the following script to generate the files:

```
$ cat > loop.tcsh << EOF
foreach ECUTWFC ( 5 10 15 20 25 30 35 40 )
  sed "s/5\.0/${ECUTWFC}/g" silicon-3.in > silicon-${ECUTWFC}.in
end
EOF
$ tcsh loop.tcsh
```

Furthermore you can use the command `grep` in order to extract the information that you are looking for automatically. For example:

```
$ grep "\!" silicon-5.out
!    total energy              =     -15.60437792 Ry
```

Exercise 2

We now want to explore one other convergence parameter of DFT calculations for crystals, the Brillouin zone sampling K_POINTS. This set of parameters specifies the grid used to discretize the Brillouin zone in reciprocal space. These concepts are discussed in Lecture **??**.

In the input file `silicon-3.in` we had requested a uniform sampling of Bloch wavevectors **k** by setting `4 4 4 1 1 1`. This means that we want to slice the Brillouin zone in a $4 \times 4 \times 4$ grid, and we shift this grid by half a grid spacing in each direction (`1 1 1`). This shift is used because it usually provides a better sampling. So now we expect the code to work with exactly $4 \times 4 \times 4 = 64$ **k**-vectors.

▶ Search for '`number of k points`' in the output file `silicon-3.in`.

You should find that the code is using only 10 **k**-points instead of the expected 64 points. The reason for this difference is that many points in our grid are equivalent by symmetry. The code recognizes these symmetries and only performs explicit calculations for the inequivalent points.

▶ Determine how the total energy of silicon varies with the number of **k**-points, using the same procedure as in Exercise 1. Consider the following situations for the input parameters
`K_POINTS: 1 1 1 0 0 0 / 2 2 2 0 0 0 / 4 4 4 0 0 0 / 8 8 8 0 0 0 / 16 16 16 0 0 0 / 32 32 32 0 0 0`.
For these calculations you can use our 'converged' cutoff `ecutwfc = 25.0` Ry.

Note. For these calculations it is convenient to use the execution flag `-npool` inside the command line, for example:

```
ibrun -n 12 pw.x -npool 4
```

This flag tells the code to distribute the **k**-points among groups of CPUs. In this case we are telling the code to use 12 CPUs, and to distribute the **k**-points in 4 groups of 12/4=3 CPUs.

Note that the number of 'pools' **cannot be smaller** than the total number of **k**-points, therefore you will not be able to use this trick for the two smallest grids `1 1 1 0 0 0` and `2 2 2 0 0 0`.

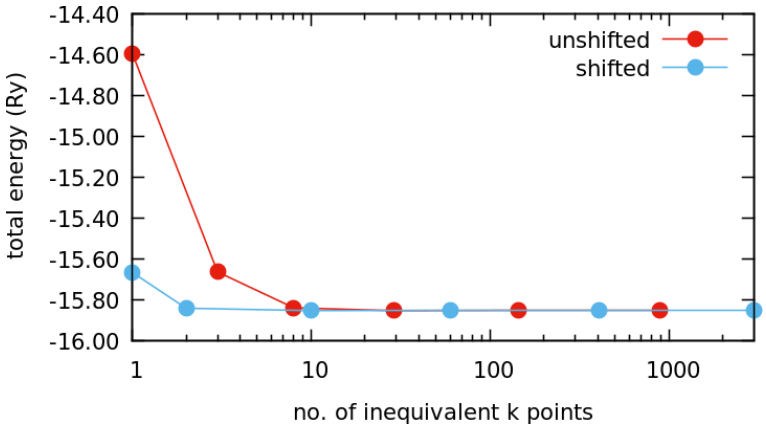
▶ Repeat the last operation, this time using nonzero shifts, eg `1 1 1 1 1 1 / 2 2 2 1 1 1 / 4 4 4 1 1 1` and so on.

You should be able to construct two files similar to the ones below:

```
$ more excercise2a.txt

#  grid     shift    energy (Ry)  inequiv. k-points
  1  1  1   0 0 0   -14.59239647     1
  ...
 32 32 32   0 0 0   -15.85108141   897

$ more excercise2b.txt

#  grid     shift    energy (Ry)  inequiv. k-points
  1  1  1   1 1 1   -15.66368659     1
  ...
 32 32 32   1 1 1   -15.85108150   2992
```

▶ Plot the total energy as a function of the number of inequivalent **k**-points in each calculation, both for the case of the unshifted grid (`0 0 0`) and the shifted grid (`1 1 1`). You should obtain something similar to the following (note the

logarithmic scale in the horizontal axis):



Here we can see that by using the `4 4 4 1 1 1` grid we obtain a total energy which is already very good, only $<$ 2 meV/atom higher than our best value at `32 32 32 1 1 1`. We also see that the shifted grid converges faster (in terms of **k**-point count and CPU time) than the unshifted grid.

► Plot the CPU time vs. the number of inequivalent **k**-points and verify that the time scales approximately linearly with the number of such points. Can you explain why this is the case?

Exercise 3

In this exercise we want to explore how the time required to perform DFT calculations scales as a function of system size.

The input file `silicon-1.in` has been modified to generate 5 new input files which you can download and unpack as follows:

```
$ cp /home1/06868/giustino/sharedfiles/HandsOn02.Ex3.tgz ./
$ tar xfz HandsOn02.Ex3.tgz ; ls HandsOn02.Ex3
```

```
silicon-3.1.in silicon-3.2.in silicon-3.3.in silicon-3.4.in silicon-3.5.in silicon-3.6.in
```

These files correspond to **supercells** of silicon, ie computational cells containing multiple primitive cells. `silicon-4.1.in` containins one primitive unit cell, `silicon-4.2.in` contains a $2 \times 2 \times 2$ supercell, and so on, up to $6 \times 6 \times 6$ primitive unit cells (`silicon-4.6.in`). By looking inside these input files you can check that we have a number of Si atoms ranging from 2 to 432.

► Now run `pw.x` using these six input files, and extract the CPU time in each case. The procedure for running jobs is the same as in the previous exercises.

Your data should look similar to the following. Note that the timing of each job depends on the number of cores, the following data were generated using 64 cores (`ibrun -n 64 pw.x`):

```
# atoms   cputime (s)
    2   0.22
   16   0.61
   54   3.35
   ...        ...
```

▶ Plot the CPU time as a function of the number of atoms. Can you identify a simple law relating these two quantities?

In general we can say that standard DFT calculations scale with the cube of the number of atoms: $T_{\rm CPU} = {\rm const} \cdot N^3$ ($N =$ number of atoms). This can be verified directly by plotting the above data using the cube of the first column: this plot should give an approximately straight line.

The take-home message of this exercise is that if we double the size of our system, then our DFT calculation will require approximately 8 times longer to complete (e.g. 1 week$\rightarrow$ 8 weeks).

Exercise 4

The motivation for using **parallel** computers is that, by splitting the computational tasks among multiple cores, we reduce the total execution time. In this exercise we want to check how the execution time depends on the number of cores used during parallel execution.

▶ Using the input file `silicon-3.3.in`, corresponding to a supercell with 54 silicon atoms, determine the CPU time required to perform run a calculation as a function of the number of cores, from 1 core to 128 cores. To this end, first you will need to execute

```
$ ibrun -n  1 pw.x < silicon-3.3.in >   1.out
$ ibrun -n  2 pw.x < silicon-3.3.in >   2.out
$ ...
$ ibrun -n 63 pw.x < silicon-3.3.in > 127.out
$ ibrun -n 64 pw.x < silicon-3.3.in > 128.out
```

Then you will need to read the CPU time in the output file, and collect the data in a table. Note that you can either run each command individually, or automate this procedure using a script similar to that described at the end of Exercise 1.

▶ Plot the **parallel speedup** factor for the previous calculation.

The parallel speedup is defined as follows:

$$\text{speedup}(n) = \frac{\text{CPU time for 1 core}}{\text{CPU time for } n \text{ cores}}$$

A hypothetical software, perfectly parallelized, should exhibit a speedup equal to the number of cores:

$$\text{ideal speedup}(n) = n$$

Plot the speedup of the previous set of calculations, and compare your data with the ideal speedup. What do you think is happening here?

Hands-On 3
Equilibrium structures

As usual we start an interactive session on LS6 and move to the `$SCRATCH` folder:

```
$ idev -m 90 -A DMR23030
$ cd $SCRATCH
```

Equilibrium structure of a diatomic molecule

In this exercise we are going to learn how to calculate the equilibrium structure of simple systems. The formal theory at the basis of these calculations is discussed in Lecture 8. Here we only need to keep in mind the following rule:

Among all possible structures, the equilibrium structure at zero temperature and zero pressure is found by minimizing the DFT total energy.

The DFT total energy is the same quantity that we have been extracting from the output files in HandsOn01 and HandsOn02, when we were issuing the command: `grep "\!" silicon-1.out`. This quantity includes all terms appearing in the electron-ion Hamiltonian, except the kinetic energy of the ions. This quantity is also called the **potential energy surface**.

Let us calculate the equilibrium structure of the Cl_2 molecule.

The Cl_2 molecule has only 2 atoms, therefore its structure is fully specified by the Cl−Cl distance. In this simple case, the determination of the equilibrium structure corresponds to finding the Cl−Cl distance with the lowest total energy.

The first step is to find a suitable pseudopotential for Cl. As in HandsOn01 we go to

http://pseudopotentials.quantum-espresso.org/legacy_tables/original-qe-pp-library

and look for Cl. We recognize LDA psedupotential by the label `pz` in the filename. Let us go for the following:

```
$ wget http://pseudopotentials.quantum-espresso.org/upf_files/Cl.pz-bhs.UPF
```

We also copy the executable and the input file from the previous tutorials:

```
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x ./
$ cp ~/PHY392Q/HandsOn01/silicon-1.in ./cl2.in
```

Now we modify the input file in order to consider the Cl_2 molecule:

```
$ more cl2.in
```

```
&control
 calculation = "scf"
 prefix = "Cl2",
 pseudo_dir  = "./",
 outdir = "./"
/
&system
 ibrav = 1,
 celldm(1) = 20.0,
```

```
 nat = 2,
 ntyp = 1,
 ecutwfc = 100,
/
&electrons
/
ATOMIC_SPECIES
 Cl  1.0  Cl.pz-bhs.UPF
ATOMIC_POSITIONS bohr
 Cl 0.0 0.0 0.0
 Cl 2.0 0.0 0.0
K_POINTS gamma
```

Using `ibrav = 1` we select a cubic simulation box of side `celldm(1)`. Here we are choosing a cubic box of side 20 bohr (1 bohr = 0.529167 Å). The keyword `gamma` means that we will be sampling the Brillouin zone only at the Γ point, that is $\mathbf{k} = 0$. This is totally fine since we want to study a molecule, not an extended crystal.

We can now execute `pw.x` in order to check that everything will run smoothly:

```
ibrun -n 24 pw.x < cl2.in > cl2.out
```

► Verify that using a planewaves cutoff of 100 Ry we achieve energy convergence to within 10 meV/atom (as compared to the most converged value that you can afford). Make a plot of the energy vs. planewaves cutoff.

Incidentally, from the output file we can see the various steps of the DFT self-consistent cycle (SCF). For example if we look for the total energy:

```
$ grep "total energy" cl2.out
```

```
     total energy              =     -55.58060868 Ry
     total energy              =     -55.80487514 Ry
     total energy              =     -55.84085668 Ry
     total energy              =     -55.84152500 Ry
     total energy              =     -55.84160081 Ry
     total energy              =     -55.84161854 Ry
     total energy              =     -55.84162030 Ry
!    total energy              =     -55.84162116 Ry
     The total energy is the sum of the following terms:
```

Here we see that the energy reaches its minimum in 8 iterations. The iterative procedure stops when the energy difference between two successive iterations is smaller than `conv_thr = 1.0d-6`. In fact if you look in the output file just below the line with the marker `!` for the total energy, you will see:

```
!    total energy              =     -55.84162116 Ry
     estimated scf accuracy    <       0.00000003 Ry
```

When we need more accurate calculations, we can specify our own convergence threshold inside the input file (in Ry), for example as follows:

```
&electrons
conv_thr = 1.d-10
```

```
/
```

Now we proceed to calculate the total energy as a function of the Cl−Cl bond length. In the reference frame chosen for the above input file, we have one Cl atom at (0,0,0) and one at (2,0,0) in atomic units. Therefore we can vary the bond length by simply displacing the second atom along the x axis.

In order to automate the procedure we can use the following script:

```
$ cat > bl.tcsh << EOF
foreach DIST ( 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6 )
  sed "s/2\.0/${DIST}/g" cl2.in > cl2_${DIST}.in
end
EOF
$ tcsh bl.tcsh
```

This script generates identical files which will differ only by the Cl−Cl bond length:

```
$ ls cl2_*
```

```
cl2_2.2.in  cl2_2.4.in  cl2_2.6.in  cl2_2.8.in  cl2_3.0.in  cl2_3.2.in
cl2_3.4.in  cl2_3.6.in  cl2_3.8.in  cl2_4.0.in  cl2_4.2.in  cl2_4.4.in
cl2_4.6.in
```

At this point we can execute `pw.x` for each of these input files:

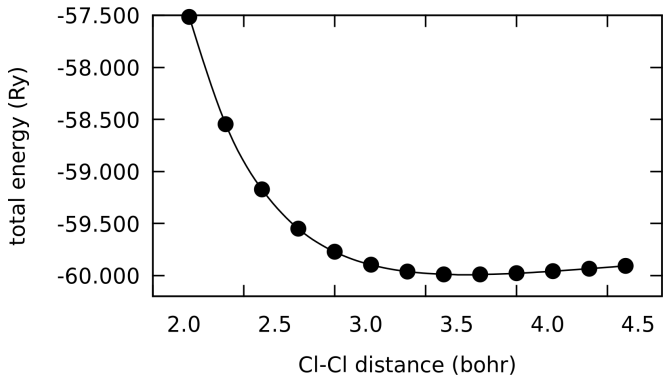
```
ibrun -n 12 pw.x < cl2_2.2.in > cl2_2.2.out
...
ibrun -n 12 pw.x < cl2_4.6.in > cl2_4.6.out
```

After running these calculations we can look for the total energy using `grep`:

```
$ grep "\!" cl2_*.out
```

```
cl2_2.2.out:!    total energy              =     -57.51390370 Ry
cl2_2.4.out:!    total energy              =     -58.54440032 Ry
...
cl2_4.6.out:!    total energy              =     -59.90658471 Ry
```

By extracting the bond length and the energy from this data we can obtain the plot shown below:



In this plot the dots are the calculated datapoints, and the line is a spline interpolation. In `gnuplot` this interpolation is obtained using the flag `smooth csplines` at the end of the `plot` command.

By zooming in the figure we find that the bond length at the minimum is 3.725 bohr = 1.97 Å. The calculated bond length is 1.5% shorter than the measured value 1.99 Å.

▶ Calculate the energy of the Cl_2 molecule as a function of bond length, as explained above, and plot your results as in the above figure. Verify that your calculated equilibrium bond length is within 1.5% of the experimental value.

▶ Repeat the previous exercise, but this time using a planewaves cutoff of 10 Ry. What can we deduce from these results?

Binding energy of a diatomic molecule

The total energy of Cl_2 at the equilibrium bond length can be used to calculate the dissociation energy of this molecule.

The dissociation energy is defined as the difference $E_{\rm diss} = E_{\rm Cl_2} - 2E_{\rm Cl}$, with $E_{\rm Cl}$ the total energy of an isolated Cl atom.

In order to evaluate this quantity, we first calculate $E_{\rm Cl_2}$ using the equilibrium bond length determined in the previous section. For this we create a new input file `cl2-2.in` by copying `cl2.in` and making the following change:

```
...
ATOMIC_POSITIONS bohr
 Cl 0.00 0.00 0.00
 Cl 3.725 0.00 0.00
...
```

A calculation with this modified input file yields the total energy:

$E_{\rm Cl_2}$ = -59.99059538 Ry

Next we consider the isolated Cl atom. The slight complication in this case is that the outermost ($3p$) electronic shell of Cl has one unpaired electron:

$$3p^5: \quad \boxed{\uparrow\downarrow}\boxed{\uparrow\downarrow}\boxed{\uparrow\ }$$

In order to take this into account we perform a **spin-polarized** calculation using the following modification of the previous input file:

```
$ cat > cl.in << EOF
&control
 calculation = "scf"
 prefix = "Cl",
 pseudo_dir = "./",
 outdir = "./"
/
 &system
 ibrav = 1,
 celldm(1) = 20.0,
 nat = 1,
```

```
 ntyp = 1,
 ecutwfc = 100,
 nspin = 2,
 tot_magnetization = 1.0,
 occupations = "smearing",
 degauss = 0.001,
/
&electrons
/
ATOMIC_SPECIES
 Cl 1.0 Cl.pz-bhs.UPF
ATOMIC_POSITIONS
 Cl 0.00 0.00 0.00
K_POINTS gamma
EOF
```

The keyword `nspin = 2` specifies that we want to consider both up-spin and down-spin electrons. The keyword `tot_magnetization = 1.0` indicates that we want 1 unpaired electron in total (i.e. all orbitals are doubly occupied except one).

After running `pw.x` with this input file, we obtain a total energy

E_{Cl} = -29.86386084 Ry

By combining the last two results we find

E_{diss} = 0.262874 Ry = 3.58 eV

This result should be compared to the experimental value of 2.51 eV.

We can see that DFT/LDA overestimates the dissociation energy of Cl_2 by about 1 eV (the calculated value is 43% higher than in experiments): interatomic bonding is too strong in LDA.

Equilibrium structure of a bulk crystal

In this section we study the equilibrium structure of a bulk crystal. We consider again a silicon crystal, since we already studied the convergence parameters in HandsOn02.

Before starting we can clean up the folder by removing all files and subfolders referring to the previous example. We can also copy the pseudopotential and executable from the folder `/PHYS392Q/HandsOn01`.

```
$ cd $SCRATCH ; rm -rf *
$ cp ~/PHY392Q/q-e-qe-7.2/bin/pw.x ./
$ cp ~/PHY392Q/HandsOn01/Si.pz-vbc.UPF ./
$ cp ~/PHY392Q/HandsOn01/silicon-1.in ./silicon.in
```

We use what we learned from the convergence tests in HandsOn02 to set the correct parameters for planewaves cutoff and Brillouin-zone sampling:

```
$ cat > silicon.in << EOF
&control
```

```
 calculation = "scf"
 prefix = "silicon",
 pseudo_dir = "./",
 outdir = "./"
/
&system
 ibrav = 2,
 celldm(1) = 10.28,
 nat = 2,
 ntyp = 1,
 ecutwfc = 25.0,
/
&electrons
/
ATOMIC_SPECIES
 Si  28.086  Si.pz-vbc.UPF
ATOMIC_POSITIONS alat
 Si 0.00 0.00 0.00
 Si 0.25 0.25 0.25
K_POINTS automatic
 4 4 4 1 1 1
EOF
```

In the case of bulk crystals we often have information about the structure from XRD measurements. This information simplifies considerably the determination of the equilibrium structure. For example, in the case of silicon, the diamond structure is uniquely determined by the lattice parameter a, therefore the minimization of the total energy is really a one-dimensional optimization problem, precisely as for the Cl_2 molecule:

https://physics.nist.gov/cgi-bin/cuu/Value?asil|search_for=silicon

Later on we will explore the slightly more complicated situation where we need to decide which one among several possible crystal structures is the most stable.

To find the equilibrium lattice parameter of silicon we perform total energy calculations for a series of plausible parameters. We can generate multiple input files at once by using the following script:

```
$ cat > silicon.tcsh << EOF
foreach ALAT ( 9.95 10.0 10.05 10.1 10.15 10.2 10.25 10.3 10.35 10.4 10.45 10.5 )
sed "s/10\.28/${ALAT}/g" silicon.in > silicon_${ALAT}.in
end
EOF
```

Now we can execute `pw.x` using the generated input files:

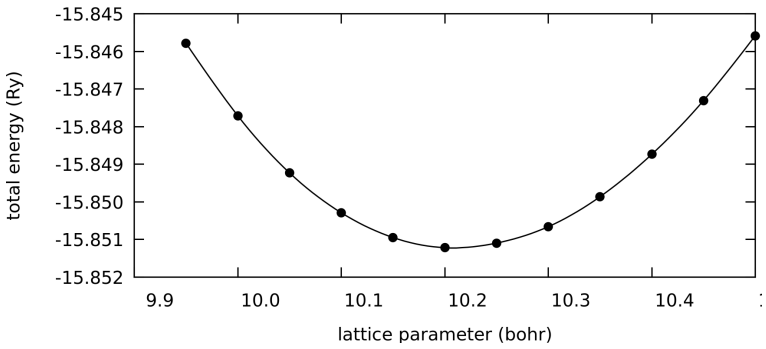
```
ibrun -n 12 pw.x < silicon_9.95.in > silicon_9.95.out
...
ibrun -n 12 pw.x < silicon_10.5.in > silicon_10.5.out
```

At the end of the execution we should be able to see the output files `silicon_9.95.out`, $\cdots$ `silicon_10.5.out`, and

we can extract the total energies as follows:

```
$ grep "\!" silicon_*.out
silicon_9.95.out:!    total energy              =     -15.84578278 Ry
...
silicon_10.5.out:!    total energy              =     -15.84558106 Ry
```

► Determine the energies of silicon as a function of the lattice parameter, and plot your results. Find the **equilibrium lattice parameter**, that is the lattice parameter with the lowest energy. Your plot show look like the following:



Also in this case the dots are the calculated datapoints, and the line is a smooth interpolating function (obtained using '`smooth csplines`' in `gnuplot`).

By zooming near the bottom we see that the equilibrium lattice parameter is a = 10.2078 bohr = 5.4016 Å. Our calculated value is very close to the measured equilibrium parameter of 5.43 Å; DFT/LDA **underestimates** the measured lattice constant by 0.5%.

Cohesive energy of a bulk crystal

The **cohesive energy** is defined as the heat of sublimation of a solid into its elements. This quantity is the solid-state counterpart of the molecular dissociation energy that we have seen for Cl_2. It quantifies how much energy is needed to break a solid into a set of isolated atoms.

The calculation is almost identical to the case of the dissociation energy of the Cl_2 molecule: we need to take the difference between the total energy at the equilibrium lattice parameter and the total energy of each atom in isolation.

For the energy at equilibrium we just repeat a calculation from the previous exercise, but this time using the equlibrium lattice parameter just determined:

```
...
 celldm(1) = 10.2078,
...
```

This calculation yields:

$E_{\rm bulk}$ = -15.85122368 Ry

(this is the total energy per unit cell, and each unit cell contains 2 Si atoms)

To determine the total energy of one Si atom in isolation we consider a cubic box with a large lattice parameter and one Si atom, as we did for the Cl atom.

In this case we need to consider that Si has 4 valence electrons in the configuration $3s^2p^2$. According to Hund's rules the spins in the p shell must be arranged as follows:

$3p^2$: 

We can modify the input file as follows:

```
$ cat > si.in << EOF
&control
 calculation = "scf",
 prefix = "silicon",
 pseudo_dir = "./",
 outdir = "./"
/
&system
 ibrav = 1,
 celldm(1) = 20.0,
 nat = 1,
 ntyp = 1,
 ecutwfc = 25.0,
 nspin = 2,
 tot_magnetization = 2.0,
 occupations = "smearing",
 degauss = 0.001
/
&electrons
/
ATOMIC_SPECIES
Si 28.086 Si.pz-vbc.UPF
ATOMIC_POSITIONS alat
Si 0.00 0.00 0.00
K_POINTS gamma
EOF
```

Here the input variable `tot_magnetization = 2.0` is used to request that the code finds the lowest-energy electronic configuration with two electron spins pointing in the same direction, and all other electrons occupy the remaining orbitals in pairs $\uparrow\downarrow$.

The calculation for the isolated atom gives:

$E_{\rm Si}$ = -7.53721939 Ry

By combining the last two results we obtain:

$E_{\rm cohes} = (E_{\rm bulk} - 2E_{\rm Si})/2$ = 0.38839 Ry = 5.28 eV

Note that we divide by 2 since there are 2 Si atoms in each crystal unit cell, so that this quantity gives the **cohesive energy per atom**.

The measured heat of sublimation of silicon is 4.62 eV, therefore our DFT/LDA calculation overestimates the experimental value by 14%.

The **underestimation of lattice parameters** and the **overestimation of cohesive energies** are typical of DFT/LDA. We can summarize these observations by stating that **DFT/LDA tends to overbind** molecules and solids.

Note While the DFT/LDA tends to overbind, another extremely popular approximation to the exchange and correlation functional, the PBE functional [Perdew, Burke, Ernzerhof, PRL 77, 3865 (1996)] tends to **underbind**. For example, DFT/PBE usually yields lattice parameter slightly larger than in experiments ($\sim$1%).

► Familiarize yourself with all the steps of this Hands-On session, and generate data tables and plots as requested for each exercise.

Hands-On 4
Layered systems and visualization

We start an interactive session on LS6 and move to the `$SCRATCH` folder:

```
$ idev -m 90 -A DMR23030
$ cd $SCRATCH
```

In this hands-on session we will study the equilibrium structure of simple crystals, namely diamond and graphite.

Equilibrium structure of diamond

The crystal structure of diamond is almost identical to the one that we used for silicon in HandsOn03. The two important differences are (i) this time we need a pseudopotential for diamond, and (ii) we expect the equilibrium lattice parameter to be considerably smaller than in silicon.

▶ Find a suitable LDA pseudopotential for diamond. I recommend to use the pseudopotential `C.pz-vbc.UPF`, but any other choice will do as well. The link to the pseudopotential library can be found in HandsOn03.

▶ Download this pseudopotential, and copy the executable `pw.x` into the current directory. Create an input file for diamond, `diamond.in`, by modifying the input file for silicon from HandsOn03. For the time being we can set the lattice parameter to the experimental value, 3.56 Å.

▶ Execute `pw.x` to make sure that everything goes smoothly.

▶ Calculate the total energy as a function of planewaves cutoff `ecutwfc`, and plot your data.

You can generate the input files for various cutoff energies manually, or by using the following script (adapted from HandsOn3):

```
$ cat > loop-diamond.tcsh << EOF
foreach ECUTWFC ( 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 150 200 300 400)
  sed "s/25\.0/${ECUTWFC}/g" diamond.in > diamond-${ECUTWFC}.in
end
EOF
$ tcsh loop-diamond.tcsh
```

▶ Determine the planewaves cutoff that is required to have the total energy per atom converged to within 10 meV. We can take as 'exact' result the value for the highest cutoff considered, 400 Ry.

▶ Using the planewaves cutoff determined in the previous step, calculate the total energy of diamond as a function of lattice parameter and plot your data.

▶ Determine the equilibrium lattice parameter of diamond, and compare your result with the experimental value. Do your calculations overestimate or underestimate the experimental lattice parameter?

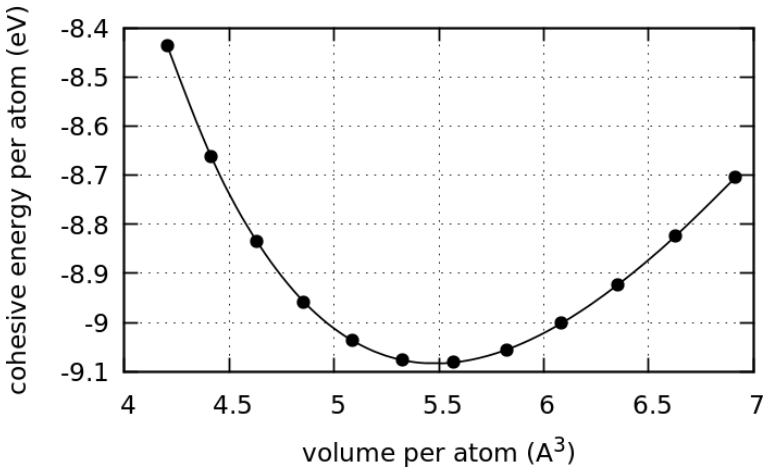
▶ Using the equilibrium lattice parameter determined in the previous exercise, calculate the cohesive energy of diamond and compare your value with experiments. For this calculation we use the same strategy employed in HandsOn03 for silicon. The C atom in its ground state has a valence electronic configuration $2s$ $\boxed{\uparrow\downarrow}$ $2p$ $\boxed{\uparrow}$ $\boxed{\uparrow}$ $\boxed{}$, therefore we need to run a spin-polarized calculation.

▶ Compare your calculated cohesive energy of diamond with the experimental value, 7.37 eV.

▶ Plot the cohesive energy vs. volume/atom for all the lattice parameters that you considered. Use units of eV for the energy, and Å^3 for the volume. The volume of the unit cell in each calculation can be found in the output file, next to the keyword

`unit-cell volume`

As a reference, the plot should like like the following.



Equilibrium structure of graphite

In this exercise we study the equilibrium structure of graphite. We consider the structure of graphite in the Bernal stacking (AB), as obtained from solid state physics textbooks or online databases (we will discuss databases later in the course):

https://en.wikipedia.org/wiki/Graphite

The unit cell of graphite is hexagonal, with lattice vectors

$$\begin{array}{lccc} \mathbf{a}_1 = a\,(& 1 & 0 & 0 \quad) \\ \mathbf{a}_2 = a\,(& -1/2 & \sqrt{3}/2 & 0 \quad) \\ \mathbf{a}_3 = a\,(& 0 & 0 & c/a \quad) \end{array}$$

(a =2.464 Å and c/a = 2.724), and with 4 C atoms per primitive unit cell, with fractional coordinates:

$$\begin{array}{lcccc} C_1 : (& 0 & 0 & 1/4 &) \\ C_2 : (& 0 & 0 & 3/4 &) \\ C_3 : (& 1/3 & 2/3 & 1/4 &) \\ C_4 : (& 2/3 & 1/3 & 3/4 &) \end{array}$$

▶ Starting from the input file that you used for diamond in the previous exercise, build an input file for calculating the total energy of graphite, using the experimental crystal structure given above.

Here you will need to pay attention to the entries `ibrav` and `celldm()` in the input file. Search for these entries in the documentation page:

https://www.quantum-espresso.org/Doc/INPUT_PW.html

Here you should find the following:

ibrav INTEGER

Status: REQUIRED

```
Bravais-lattice index. Optional only if space_group is set.
If ibrav /= 0, specify EITHER [ celldm(1)-celldm(6) ]
OR [ A, B, C, cosAB, cosAC, cosBC ]
but NOT both. The lattice parameter "alat" is set to
alat = celldm(1) (in a.u.) or alat = A (in Angstrom);
see below for the other parameters.
For ibrav=0 specify the lattice vectors in CELL_PARAMETERS,
optionally the lattice parameter alat = celldm(1) (in a.u.)
or = A (in Angstrom). If not specified, the lattice parameter is
taken from CELL_PARAMETERS
IMPORTANT NOTICE 1:
with ibrav=0 lattice vectors must be given with a sufficiently large
number of digits and with the correct symmetry, or else symmetry
detection may fail and strange problems may arise in symmetrization.
IMPORTANT NOTICE 2:
do not use celldm(1) or A as a.u. to Ang conversion factor,
use the true lattice parameters or nothing,
specify units in CELL_PARAMETERS and ATOMIC_POSITIONS

ibrav      structure                  celldm(2)-celldm(6)
                                   or: b,c,cosbc,cosac,cosab
  0          free
      crystal axis provided in input: see card CELL_PARAMETERS

  1          cubic P (sc)
      v1 = a(1,0,0),  v2 = a(0,1,0),  v3 = a(0,0,1)

  2          cubic F (fcc)
      v1 = (a/2)(-1,0,1),  v2 = (a/2)(0,1,1), v3 = (a/2)(-1,1,0)

  3          cubic I (bcc)
      v1 = (a/2)(1,1,1),  v2 = (a/2)(-1,1,1),  v3 = (a/2)(-1,-1,1)
 -3          cubic I (bcc), more symmetric axis:
      v1 = (a/2)(-1,1,1), v2 = (a/2)(1,-1,1),  v3 = (a/2)(1,1,-1)

  4          Hexagonal and Trigonal P        celldm(3)=c/a
      v1 = a(1,0,0),  v2 = a(-1/2,sqrt(3)/2,0),  v3 = a(0,0,c/a)
```

Based on this information we must use `ibrav = 4` and set `celldm(1)` to a, `celldm(3)` to c/a.

Since we have the atomic coordinates in fractional units of $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$, we must specify the keyword `crystal` in the input file:

```
ATOMIC_POSITIONS crystal
```

As a sanity check, if you run a calculation with `ecutwfc = 100` and `K_POINTS gamma` you should obtain a total energy of -44.58184546 Ry.

► After performing convergence test with respect to the number of **k**-points, I found that the total energy is converged to 4 meV/atom when using a shifted $6 \times 6 \times 2$ grid, that is:

```
...
K_POINTS automatic
6 6 2 1 1 1
```

Using this setup for the Brillouin-zone sampling, calculate the lattice parameters of graphite a and c/a at equilibrium. Note that this requires a minimization of the total energy in a **two-dimensional** parameter space.

For this calculation it is convenient to automatically generate input files as follows, assuming that your input file is called `graphite.in`:

▶ Replace the values of `celldm(1)` and `celldm(3)` by the placeholders `ALAT` and `RATIO`, respectively;

▶ Create a script `graphite.tcsh` with the following content:

```
foreach A ( 4.4 4.5 4.6 4.7 4.8 4.9 5.0 )
  foreach CBYA ( 2.50 2.55 2.60 2.65 2.70 2.75 2.80 2.85 2.90 )
    sed "s/ALAT/${A}/g" graphite.in > tmp
    sed "s/RATIO/${CBYA}/g" tmp > graphite_${A}_${CBYA}.in
    ibrun -n 24 pw.x -npool 12 < graphite_${A}_${CBYA}.in > graphite_${A}_${CBYA}.out
  end
end
```

▶ By running `tcsh graphite.tcsh` you will be able to generate input files for all these combinations of a and c/a, execute `pw.x` for each file, and store the output in the corresponding `.out` files. Note that this will produce $7 \times 9 = 63$ input files, but the total execution time should be no more than a few minutes.

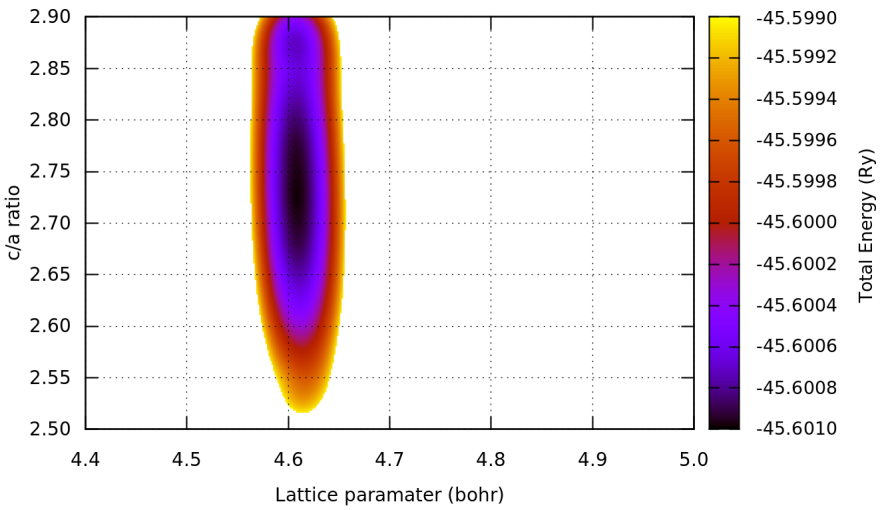
▶ the end you will be able to extract the total energies by using

```
grep "!" graphite_*_*.out > graphite.txt
```

▶ Visualize the total energy of graphite as a function of the parameters a and c/a in a 3D plot, and identify the values of the lattice parameters at equilibrium.

▶ Compare your calculated lattice parameters with the experimental values and state the % deviation.

Your plot of the total energy as a function of a and c/a should look like the following:



This plot was generated using the following commands in `gnuplot` (the file `graphite.txt` must first be cleaned up in order to obtain only three columns with the values of a, c/a, and energy):

```
set dgrid3d splines 100,100
set pm3d map
splot  [] [] [:-45.599] "graphite.txt"
```

The 'splines' keyword provides a smooth interpolation between our discrete set of datapoints. The plotting range along the energy axis is restricted in order to highlight the location of the energy minimum.

Here we see that the energy minimum is very shallow along the direction of the c/a ratio, while it is very deep along the direction of the lattice parameter a. This corresponds to the intuitive notion that the bonding in graphite is very strong within the carbon planes, and very weak in between planes.

From these calculations we can see that the agreement between DFT/LDA and experiments for the structure of graphite is excellent. This result is somewhat an artifact: most DFT functionals generally overestimate the interlayer distance in graphite due to the inadequate description of van der Waals correlation effect; since LDA generally tends to overbind (as we have seen in all examples considered so far), this overbinding compensates for the lack of van der Waals interactions, and the cancellation of errors leads to a good agreement with experiment.

Diamond vs. graphite

▶ Calculate the cohesive energy per atom of diamond and graphite, using the optimized lattice parameters as determined in the previous exercises.

▶ Based on your calculations, which carbon allotrope is more stable at ambient conditions, diamond or graphite?

▶ Compare your result with experiments by looking up online the cohesive energies of diamond and graphite, e.g. from Shin et al., J. Chem. Phys. 140, 114702 (2014).

How to visualize crystal structures

In this excercise we want to see how the structure of graphite that we are using in our input file looks like in a ball-and-stick model.

The software `xcryden` can import QE input files and visualize the atomistic structures. General info about this project can be found at `http://www.xcrysden.org`.

To use `xcryden` on LS6, we first download and unpack the archive file by typing:

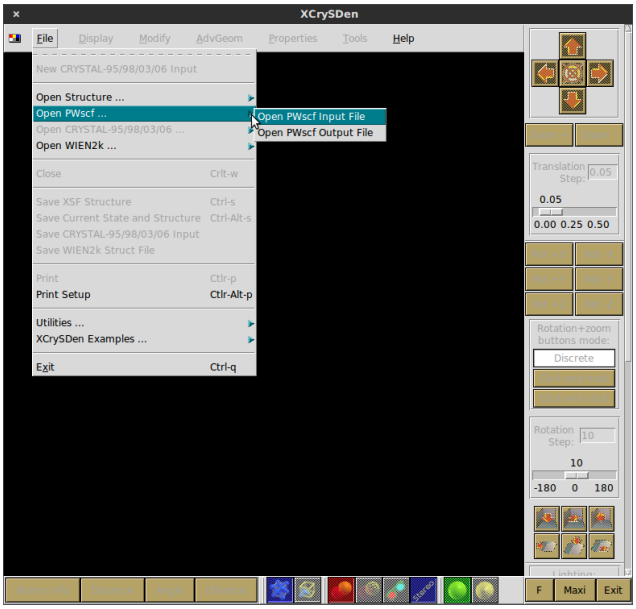
```
$ cd $HOME
$ wget http://www.xcrysden.org/download/xcrysden-1.5.60-linux_x86_64-semishared.tar.gz
$ tar xfs xcrysden-1.5.60-linux_x86_64-semishared.tar.gz
```

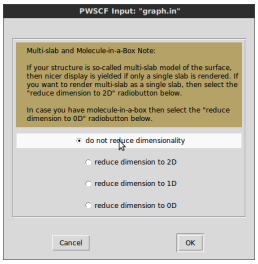
Then we can execute the program by typing (within an `idev` session):

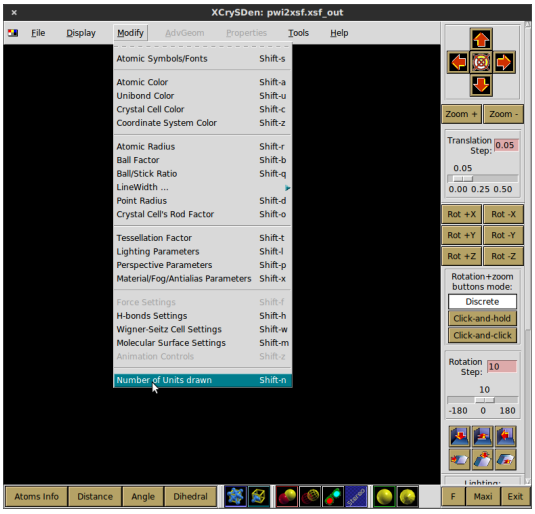
```
$ $HOME/xcrysden-1.5.60-bin-semishared/xcrysden
```

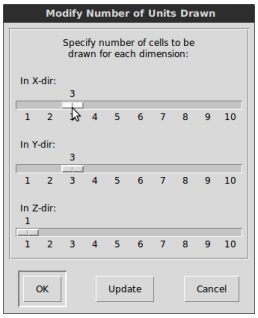
If the execution of `xcryden` on LS6 is slow, then you might want to use the code directly on your laptop. The procedure to obtain this program on your laptop is the same as that for LS6 above.

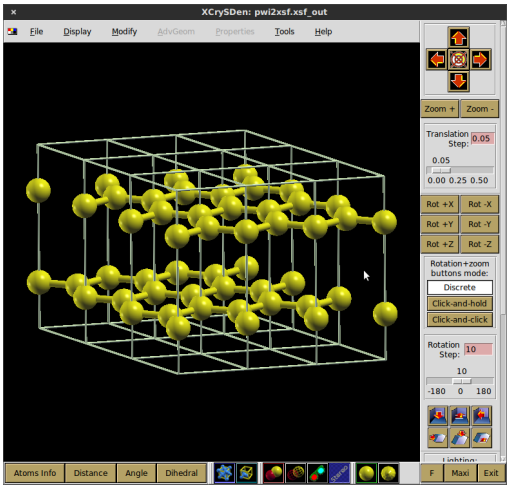
The user interface of `xcryden` is very simple and intuitive. The following snapshots will help you to get started with the visualization.











Vesta

Another software that you might want to consider for rendering structures is Vesta. In order to install Vesta on your desktop/laptop, you can visit the following page:

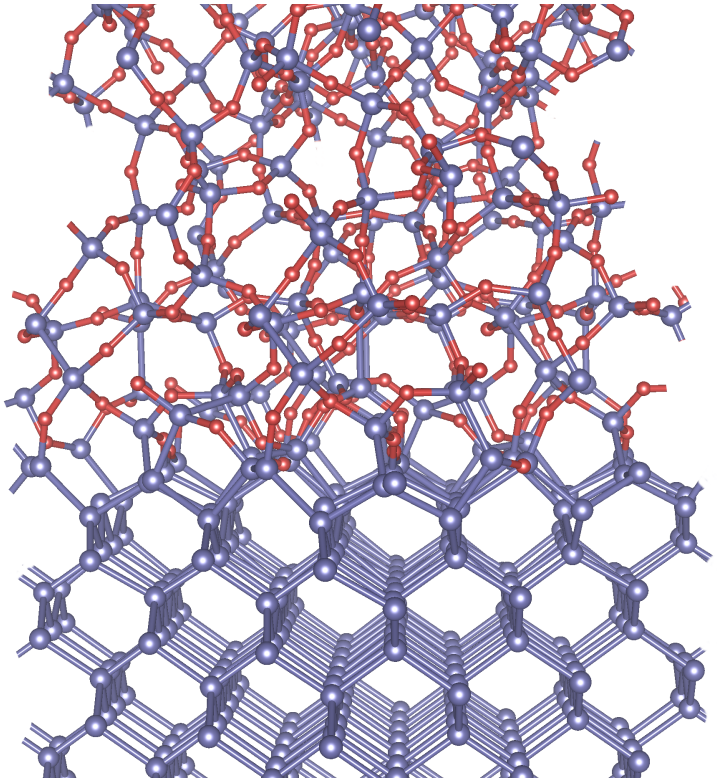
https://jp-minerals.org/vesta/en/download.html

Once installed, in order to load any structure, you can follow this sequence of steps:

- Open your structure using XcrysDen, as explained earlier
- Go to the drop-down menu
  File → Save XSF Structure
- Save your file in XSF format (this is XcrysDen native format)
- Open Vesta and load the XSF file: File → Open

The rendering provided by Vesta is usually quite impressive, for example this is the image that you would obtain for a model of the interface between crystalline silicon and its amorphous oxide [Pasquarello, Hybertsen, Car, Nature 396,

58 (1998), https://doi.org/10.1038/23908] You can find this structure on Canvas as `si-sio2.vesta`:



If you want to use VESTA directly on LS6, you can call it using the following command wihin an `idev` session:

```
$ /home1/06868/giustino/sharedfiles/VESTA-gtk3/VESTA
```

Hands-On 5
Automatic structure optimization

In this session we will learn how to find the ground-state structure using automatic optimization of atomic coordinates and crystal lattice in Quantum ESPRESSO.

Automatic optimization of atomic coordinates

In HandsOn 3 we discussed how to calculate the potential energy surface of a molecule, and how to determine equilibrium structures by locating the minima of that surface.

In this section we consider an alternative route for finding the equilibrium geometry of Cl_2. In particular, we instruct Quantum ESPRESSO to find the minimum-energy structure via the Hellman-Feynman forces.

To begin with we copy over the input files that we used for the exercise on Cl_2 in HandsOn 3:

```
$ cd $SCRATCH
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x ./
$ cp ~/PHY392Q/HandsOn03/cl2/cl2.in ./
$ cp ~/PHY392Q/HandsOn03/cl2/Cl.pz-bhs.UPF ./
```

We check that everything is in place and that everything goes smoothly by performing a test run:

```
$ ibrun -n 4 pw.x < cl2.in > cl2.out
```

If everything goes well, we should find `cl2.out` in the current folder, indicating that the run completed successfully.

Now let us take a look at the documentation of `pw.x`. As a reminder we need to go to:

https://www.quantum-espresso.org/Doc/INPUT_PW.html

We look for the input variable `calculation`; we should find the following:

Namelist: &CONTROL

calculation	CHARACTER
Default:	'scf'

A string describing the task to be performed. Options are:

- 'scf'
- 'nscf'
- 'bands'
- 'relax'
- 'md'
- 'vc-relax'
- 'vc-md'

(vc = variable-cell).

Until now we have used only one type of calculation, namely `calculation = 'scf'`. This means that we required the code to perform only a self-consistent DFT calculation with the ions **clamped** at the coordinates specified below the

keyword ATOMIC_POSITIONS.

Another possibility is to set this variable to relax inside cl2.in:

```
&control
 calculation = 'relax'
 prefix = 'cl2',
 ...
```

This choice instructs pw.x to automatically determine the equilibrium structure, starting from the coordinates given below the keyword ATOMIC_POSITIONS. In practice the code calculates the **forces** acting on the ions, and updates the ionic positions in such a way as to minimize those forces. The equilibrium configuration will correspond to the situation where all forces are **smaller than a given threshold**, and where the total potential energy has changed less than a given threshold with respect to the previous iteration.

The threshold on the forces is given by the variable forc_conv_thr:

forc_conv_thr	REAL
Default:	1.0D-3

Convergence threshold on forces (a.u) for ionic minimization:
the convergence criterion is satisfied when all components of
all forces are smaller than forc_conv_thr.
See also etot_conv_thr - both criteria must be satisfied

The threshold on the total potential energy is specified by the variable etot_conv_thr:

etot_conv_thr	REAL
Default:	1.0D-4

Convergence threshold on total energy (a.u) for ionic
minimization: the convergence criterion is satisfied
when the total energy changes less than etot_conv_thr
between two consecutive scf steps. Note that etot_conv_thr
is extensive, like the total energy.
See also forc_conv_thr - both criteria must be satisfied

In principle we could perform calculations without specifying these parameters; in this case pw.x will use some preset default values. From the above boxes we see that the default thresholds are 10^{-3} Ry/bohr for the forces, and 10^{-4} Ry for the total energy. To be more cautious, let us specify some rather stringent criteria in the input file cl2.in:

```
&control
 calculation = 'relax'
 prefix = 'cl2',
 forc_conv_thr = 1.d-5,
 etot_conv_thr = 1.d-8,
 ...
```

When we perform the automatic optimization of the atomic coordinates we also need to add a 'card' for the ions, as follows:

```
...
&electrons
/
&ions
/
ATOMIC_SPECIES
..
```

This extra card is to specify runtime parameters, such as the method used to update the atomic positions at each iteration. We can leave this card empty for the time being.

As a starting point for the atomic coordinates let us use a very large Cl-Cl separation:

```
...
ATOMIC_POSITIONS bohr
 Cl 0.00 0.00 0.00
 Cl 5.00 0.00 0.00
...
```

We now execute `pw.x` and look at the output file `cl2.out` directly using `vi`.

Note that, if you want to follow the progress of the calculation as `cl2.out` is being written, you can use the following trick:

```
$ ibrun -n 8 pw.x < cl2.in > cl2.out &
$ tail -f cl2.out
```

In these expressions the ampersand `&` tells the system to execute in the background the command that preceeds it, so we can still use the terminal while `pw.x` is being executed. The command `tail` shows you the last 10 lines of any text file. By using the flag `-f` this command 'follows' the file, so it keeps printing the last 10 lines whenever the file is modified.

In order to search for a word in `vi` we simply press `/` and type the word. We search for '`Forces`' and obtain:

```
     Forces acting on atoms (cartesian axes, Ry/au):

     atom    1 type  1   force =     0.12436001    0.00000000    0.00000000
     atom    2 type  1   force =    -0.12436001    0.00000000    0.00000000
```

These lines are telling us that, in the initial configuration, the two Cl atoms experience forces directed along the Cl-Cl axis, and pointing towards the other Cl atom. This was to be expected since we started with the atoms at a distance much larger than the equilibrium bond length.

Immediately below the forces we see the **updated** atomic positions which will be used at the next iteration:

```
ATOMIC_POSITIONS (bohr)
Cl            0.1243600093        0.0000000000        0.0000000000
Cl            4.8756399907        0.0000000000        0.0000000000
```

Clearly the atoms are being displaced towards each other. At the end of the iterations we can see something like the following:

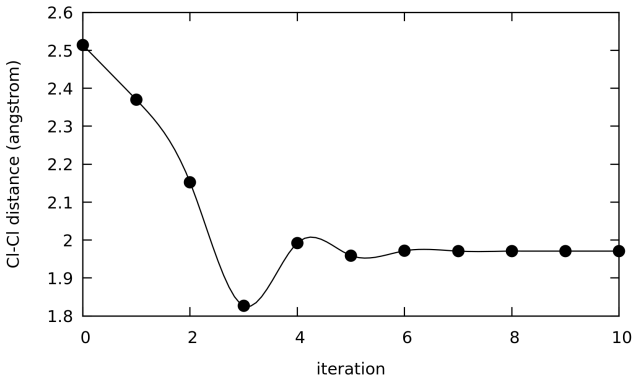
```
     Forces acting on atoms (cartesian axes, Ry/au):

     atom    1 type  1   force =    -0.00002397    0.00000000    0.00000000
     atom    2 type  1   force =     0.00002397    0.00000000    0.00000000
...
     Final energy   =     -59.9905957145 Ry
...
Begin final coordinates

ATOMIC_POSITIONS (bohr)
Cl            0.6377132745        0.0000000000        0.0000000000
Cl            4.3622867255        0.0000000000        0.0000000000
End final coordinates
```

Here we see that the forces are practically vanishing, therefore we reached the equilibrium configuration. The total energy at equilibrium is -59.9905957145 Ry, and the bond length is 3.72457 bohr = 1.97 Å. These values are in agreement with what we had found in HandsOn 3 by explicitly looking for the minimum of the potential energy surface.

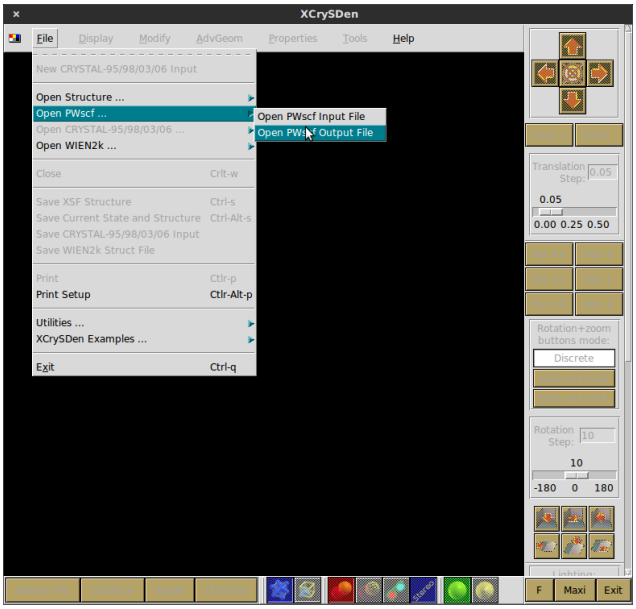
If we want to see how the atomic coordinates evolved towards the equilibrium configuration, we can simply issue:

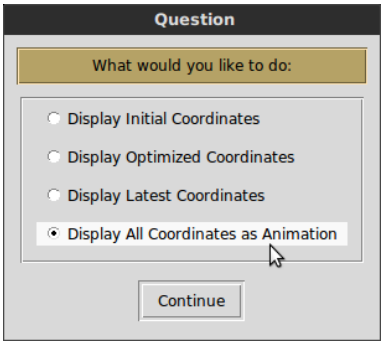
```
$ grep "Cl" cl2.out
```

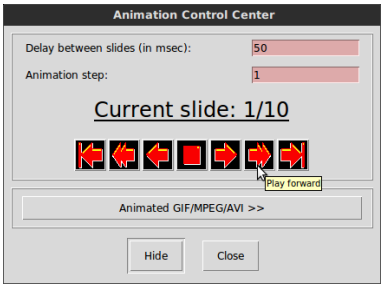
► From the atomic positions at each iteration, determine the Cl-Cl distance in Å and plot the distance vs. iteration number. You should obtain something like the following:



There is also a more visual way to examine the evolution of the atomic coordinates: we can open `XcrysDen` and go through the following steps:







Automatic optimization of atomic coordinates and unit cell

In addition to the optimization of atomic coordinates, it is also possible to optimize the vectors of the primitive unit cell. This feature is activated in `pw.x` by setting the calculation type to `vc-relax`:

Namelist: &CONTROL

calculation	CHARACTER
Default:	'scf'

A string describing the task to be performed. Options are:

'scf'

'nscf'

'bands'

'relax'

'md'

'vc-relax'

'vc-md'

(vc = variable-cell).

Let us consider the case of **graphite** as in HandsOn 4. We can copy over the corresponding input file and the carbon pseudopotential:

```
$ cp ~/PHY392Q/HandsOn04/graphite/C.pz-vbc.UPF ./
$ cp ~/PHY392Q/HandsOn04/graphite/graphite.in ./
```

Let us modify the input file is such a way as to start from a highly-compressed unit cell of graphite:

```
&control
 calculation = 'vc-relax'
 prefix = 'graphite',
 pseudo_dir = './',
 outdir = './'
/
&system
 ibrav = 4,
 celldm(1) = 4.0,
 celldm(3) = 2.0,
 nat = 4,
 ntyp = 1,
 ecutwfc = 100,
/
&electrons
/
&ions
/
&cell
/
ATOMIC_SPECIES
```

```
 C 1.0 C.pz-vbc.UPF
ATOMIC_POSITIONS crystal
C 0 0 1/4
C 0 0 3/4
C 1/3 2/3 1/4
C 2/3 1/3 3/4
K_POINTS  automatic
 6 6 2 1 1 1
```

In the above input file we should note the 'cards' `&ions` and `&cell` which are required when running this kind of calculations. In this file the cards are left empty; generally they can be used to fine-tune the optimization procedure.

When instructed to execute a calculation of type `vc-relax`, `pw.x` evaluates the **stress tensor** of the system in the initial configuration, and **updates the unit cell vectors in such a way as to reduce the stress**, similarly to what happens for the atomic forces.

Let us execute `pw.x` using the above input file. At the end of the run we can look inside the output file using `vi` and search for the following words:

`/` `stress`

We will see something like:

```
     Computing stress (Cartesian axis) and pressure

          total   stress  (Ry/bohr**3)                   (kbar)     P=     4044.41
   0.03445714   0.00000000   0.00000000         5068.82        0.00         0.00
   0.00000000   0.03445714   0.00000000            0.00     5068.82         0.00
   0.00000000   0.00000000   0.01356567            0.00        0.00      1995.58
```

This indicates that, as expected, in the first iteration the system is under very high pressure, precisely 4.04 Mbar. The sign convention is such that a positive value indicates compressive stress. Following this initial iteration, `pw.x` modifies the lattice vectors in the direction of lower pressure.

We can note that in this case the **forces** acting on each atom are all vanishing by symmetry:

```
     Forces acting on atoms (cartesian axes, Ry/au):
     atom    1 type  1   force =     0.00000000    0.00000000    0.00000000
     atom    2 type  1   force =     0.00000000    0.00000000    0.00000000
     atom    3 type  1   force =     0.00000000    0.00000000    0.00000000
     atom    4 type  1   force =     0.00000000    0.00000000    0.00000000
```

As a result this procedure will not modify the Wickoff positions of the 4 C atoms in the unit cell. The final optimized structure is found by looking for

`/` `Begin final coordinates`

```
Begin final coordinates
     new unit-cell volume =    200.60018 a.u.^3 (    29.72588 Ang^3 )
     density =      0.22345 g/cm^3
```

```
CELL_PARAMETERS (alat=  4.00000000)
   1.149455589   0.000000000  -0.000000000
  -0.574727794   0.995457740  -0.000000000
  -0.000000000  -0.000000000   2.739279264

ATOMIC_POSITIONS (crystal)
C            -0.0000000000       -0.0000000000        0.2500000000
C             0.0000000000        0.0000000000        0.7500000000
C             0.3333333333        0.6666666667        0.2500000000
C             0.6666666667        0.3333333333        0.7500000000
End final coordinates
```

In this output file we should note that the lattice vectors are given in units of the original lattice parameter in input, that is `alat = 4.0` bohr. Therefore the optimized lattice parameter is now

$a = 4.00000000 \cdot 1.149455589 = 4.598\,\text{bohr} = 2.433\,\text{Å}$

The optimized c/a ratio is

$c/a = 2.739279264/1.149455589 = 2.383.$

It is immediate to see that the lattice vectors correspond to an hexagonal lattice; for example the second line of `CELL_PARAMETERS` is $a(-1/2, \sqrt{3}/2, 0)$.

The total energy in the optimized configuration is -45.59330381 Ry.

Note. From this calculation we have obtained a c/a ratio which is much smaller than the one determined in HandsOn 4 by studying the potential energy surface (2.383 here vs. 2.729 in HandsOn 4). The interlayer separation is approximately 15% shorter in the present calculation. This result is a calculation artifact. What is happening here is that the code modifies the crystal structure so as to minimize the energy. Now, the Hamiltonian describing the system is expressed in a basis of planewaves, and the wavevectors of these planewaves along the c axis are multiples of $2\pi/c$. If, during the optimization, the c parameter undergoes a significant change (as it is the case here, since we start from $c/a = 2$ and we end up with $c/a = 2.729$), then we are effectively reducing our planewaves cutoff at each iteration. As a result the calculation becomes less and less accurate. To avoid providing results for a modified cutoff, `pw.x` performs **one additional calculation** at the very end, after having regenerated all reciprocal lattice vectors according to the optimized structure. We can see that this step yields a residual pressure of 92.9 kbar along the c-axis:

```
     Computing stress (Cartesian axis) and pressure

          total   stress  (Ry/bohr**3)                   (kbar)     P=       51.64
   0.00021089   0.00000000  -0.00000000          31.02        0.00       -0.00
   0.00000000   0.00021089   0.00000000           0.00       31.02        0.00
  -0.00000000   0.00000000   0.00063134          -0.00        0.00       92.87
```

This indicates that the structure is not yet fully optimized. In order to avoid this problem we should run a new calculation, starting from the latest lattice parameters.

▶ Perform a structural optimization for graphite once again, this time starting from the lattice parameters $a = 2.433$ Å and $c/a = 2.383$ obtained at the previous step.

▶ Compare your new optimized lattice parameter with the results of HandsOn 4. What can we deduce from this

comparison? How do we decide which optimized structure is the correct one?

Hands-On 6
Elastic constants

In this session we will learn how to calculate elastic constants. We will start with the simple case of **diamond**, and then we will try to set up an entirely new calculation on **$SrTiO_3$** (STO). For STO we will use the Materials Project database to find the initial geometry.

Elastic constants of diamond

We want calculate the elastic constants of diamond. As we have seen in Lecture 9 (Sec. 9.4), for a cubic system like diamond there are only three independent elastic constants, namely C_{11}, C_{12}, and C_{44}.

We can determine these constants by using the relations discussed in Lecture 9, in particular Eqs. (9.18)-(9.20).

We start by copying the pseudopotential for carbon and the the input file for the optimized structure from HandsOn 4:

```
$ cd $SCRATCH
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x ./
$ cp ~/PHY392Q/HandsOn04/diamond/C.pz-vbc.UPF ./
$ cp ~/PHY392Q/HandsOn04/diamond/diamond_optimized.in ./diamond0.in
```

Isotropic deformation. The isotropic deformation of the diamond structure corresponds to a uniform scaling of the lattice vectors:

$$\mathbf{a}_i' = (1+\eta)\,\mathbf{a}_i \text{ for } i = 1,2,3$$

From Lecture 9 we know that the resulting change in the total potential energy from the equilibrium value U_0 is:

$$\frac{U-U_0}{\Omega} = \frac{3}{2}\,(C_{11}+2C_{12})\,\eta^2 \tag{6.1}$$

The calculation of U and U_0 can be performed by using the following input file for diamond, which has been adapted from HandsOn 4:

```
&control
 calculation = "relax"
 prefix = "diamond",
 pseudo_dir = "./",
 outdir = "./",
 etot_conv_thr = 1.d-5,
 forc_conv_thr = 1.d-4,
/
&system
 ibrav = 0,
 celldm(1) = 6.66407,
 nat = 2,
 ntyp = 1,
 ecutwfc = 100.0,
/
&electrons
```

```
/
&ions
/
ATOMIC_SPECIES
 C 1.0 C.pz-vbc.UPF
ATOMIC_POSITIONS alat
 C 0.00 0.00 0.00
 C 0.25 0.25 0.25
K_POINTS automatic
 6 6 6 1 1 1
CELL_PARAMETERS alat
 -0.50 0.00 0.50
  0.00 0.50 0.50
 -0.50 0.50 0.00
```

In this version of the input file we are specifying that the unit cell vectors are given manually, `ibrav = 0`; these vectors are provided in units of the lattice parameter, `celldm(1)`, after the keywoord `CELL_PARAMETERS`. The lattice parameters are always printed by `pw.x`, therefore we can find this information from the output files in HandsOn 4. In this input file we also specify `calculation = "relax"`, in order to allow atoms to find their minimum energy configuration for a given set of primitive lattice vectors. Note that we also used more stringent convergence parameters for energy and forces, since we need to take differences between the energies of very similar configurations.

By running `pw.x` with the above input file we obtain the total energy in the ground state:

$U_0 =$ -22.8016487507 Ry

Furthermore we can read the volume of the unit cell by searching for: `/` `volume`

This gives $\Omega = 73.9876$ bohr^3.

Now we can modify this input file in order to establish an isotropic deformation with $\eta = 0.002$ (let us call the new file `diamond+.in`):

```
$ more diamond+.in

...
CELL_PARAMETERS alat
 -0.501 0.000 0.501
  0.000 0.501 0.501
 -0.501 0.501 0.000
```

With this new input file we find the total energy:

$U =$ -22.8016135208 Ry

Using Eq. (6.1) with $\eta = 0.002$ we obtain

$C_{11} + 2C_{12} = 0.0793599$ Ry/bohr$^3 = 1167$ GPa 1 Ry/bohr$^3 = 14710.5$ GPa

In order to increase the accuracy of our calculation, we can repeat the calculation with $\eta = -0.002$ and averae the results. This operation is mequivalent to evaluating the second derivative of U with respect of η via the central finite-difference

formula. To this end, we modify the input file as:

```
$ more diamond-.in

...
CELL_PARAMETERS alat
 -0.499 0.000 0.499
  0.000 0.499 0.499
 -0.499 0.499 0.000
```

With this new input file we find the total energy:

U = -22.8016049497

and the relation between the elastic constants is:
$C_{11} + 2C_{12} = 0.0986674$ Ry/bohr$^3 = 1451$ GPa

The average between the forward and backward difference expressions is:

$C_{11} + 2C_{12} = (1167 + 1451)/2 = 1309$ GPa

Tetragonal deformation

From Lecture 9 we have the relation:

$$\frac{U - U_0}{\Omega} = 3\,(C_{11} - C_{12})\,\eta^2 \tag{6.2}$$

and the tetragonal distortion of the unit cell can be realized by considering an expansion $(1 + \eta)$ along x and y, and a contraction $(1 - 2\eta)$ along z. We copy the input file `diamond+.in` into `diamond+_tetra.in` and we modify this new file as follows:

```
...
CELL_PARAMETERS alat
 -0.501 0.000 0.498
  0.000 0.501 0.498
 -0.501 0.501 0.000
```

This calculation gives:

U = -22.8015911554 Ry

Using Eq. (6.2) with $\eta = 0.002$ and remembering that U_0 = -22.8016487507 Ry, we obtain:

$C_{11} - C_{12}$ = 0.0648704 Ry/bohr^3 = 954 GPa

Now we can repeat the calculation for $\eta = -0.002$. The new input file contains the lattice parameters:

```
...
CELL_PARAMETERS alat
 -0.499 0.000 0.502
  0.000 0.499 0.502
 -0.499 0.499 0.000
```

This calculation gives:

U = -22.8015897742 Ry

Using Eq. (6.2) with $\eta = -0.002$ and remembering that U_0 = -22.8016487507 Ry, we obtain:

$C_{11} - C_{12}$ = 0.064261 Ry/bohr^3 = 977 GPa

The average between the results for forward and backward difference is:

$C_{11} - C_{12} = (954 + 977)/2 = 965$ GPa

By combining the two relations just obtained for C_{12} and C_{12},

$C_{11} + 2C_{12} = 1309$ GPa
$C_{11} - C_{12} = 965$ GPa

we obtain:

$C_{11} = 1080$ GPa, $C_{12} = 115$ GPa.

The corresponding experimental values are $C_{11} = 1079$ GPa, $C_{12} = 124$ GPa, from McSkimin & Andreatch, J. Appl. Phys. 43, 2944 (1972). The calculated elastic constants are within 8% of the measured values.

Trigonal deformation. From Lecture 9 we have the relation:

$$\frac{U - U_0}{\Omega} = \frac{1}{2} C_{44} \eta^2 \tag{6.3}$$

A trigonal distortion of the unit cell can be realized by considering the following distortion of the lattice parameters, as in Lec. 9:

$$u_1 = u_2 = u_3 = u_4 = u_5 = 0, \qquad u_6 = \eta$$

We start from the undistorted lattice parameters:

$$\begin{aligned} a_{1x} &= -\frac{a}{2} \\ a_{1y} &= 0 \\ a_{1z} &= \frac{a}{2} \\ a_{2x} &= 0 \\ a_{2y} &= \frac{a}{2} \\ a_{2z} &= \frac{a}{2} \\ a_{3x} &= -\frac{a}{2} \\ a_{3y} &= \frac{a}{2} \\ a_{3z} &= 0 \end{aligned}$$

Since $u_6 = 2\epsilon_6$ and $\epsilon_6 = \epsilon_{xy} = \epsilon_{yx}$, we can write:

$$\begin{aligned} a'_{1x} &= a_{1x} + \epsilon_{xy} a_{1y} = -\frac{a}{2} \\ a'_{1y} &= a_{1y} + \epsilon_{yx} a_{1x} = -\frac{\eta}{2}\frac{a}{2} \end{aligned}$$

$$
\begin{aligned}
a'_{1z} &= a_{1z} = \frac{a}{2} \\
a'_{2x} &= a_{2x} + \epsilon_{xy} a_{2y} = \frac{\eta}{2}\frac{a}{2} \\
a'_{2y} &= a_{2y} + \epsilon_{yx} a_{2x} = \frac{a}{2} \\
a'_{2z} &= a_{2z} = \frac{a}{2} \\
a'_{3x} &= a_{3x} + \epsilon_{xy} a_{3y} = -\frac{a}{2}\left(1 - \frac{\eta}{2}\right) \\
a'_{3y} &= a_{3y} + \epsilon_{yx} a_{3x} = \frac{a}{2}\left(1 - \frac{\eta}{2}\right) \\
a'_{3z} &= a_{3z} = 0
\end{aligned}
$$

These relations indicate that, if we set $\eta = 0.002$, the input file must be modified as follows:

```
...
CELL_PARAMETERS alat
 -0.5000 -0.0005 0.5000
  0.0005  0.5000 0.5000
 -0.4995  0.4995 0.0000
```

The calculation with this input file gives:

U =-22.8016460518 Ry

Using Eq. (6.3) with $\eta = 0.002$ we obtain

C_{44} = 0.0182389 Ry/bohr^3 = 268 GPa

Now we can try the calculation with $\eta = -0.002$. The lattice parameters are modified as follows:

```
...
CELL_PARAMETERS alat
 -0.5000  0.0005 0.5000
 -0.0005  0.5000 0.5000
 -0.5005  0.5005 0.0000
```

The calculation with this input file gives:

U =-22.8016389179 Ry

Using Eq. (6.3) with $\eta = -0.002$ we obtain

C_{44} = 0.066449 Ry/bohr^3 = 977 GPa

The average between the results for forward and backward difference is:

$C_{44} = (268 + 977)/2 = 622$ GPa

The corresponding experimental value is $C_{44} = 578$ GPa [McSkimin & Andreatch, J. Appl. Phys. 43, 2944 (1972)].

▶ Calculate the elastic constants C_{11}, C_{12}, and C_{44} of diamond, by following the steps illustrated above. Determine the elastic constants using two different sets of calculations: one for $\eta = 0.002$, and one for $\eta = 0.001$. Which one is more accurate? Why?

The bulk modulus B is a measure of the resistance of a materials to hydrostatic compression. This quantity can be obtained from the elastic constants as:

$$B = \frac{1}{3}(C_{11} + 2C_{12}).$$

▶ Calculate the bulk modulus of diamond, using the data obtained in the previous step, and compare your result with experimental data from the literature. Indicate the reference to the literature paper from which you obtained the experimental data.

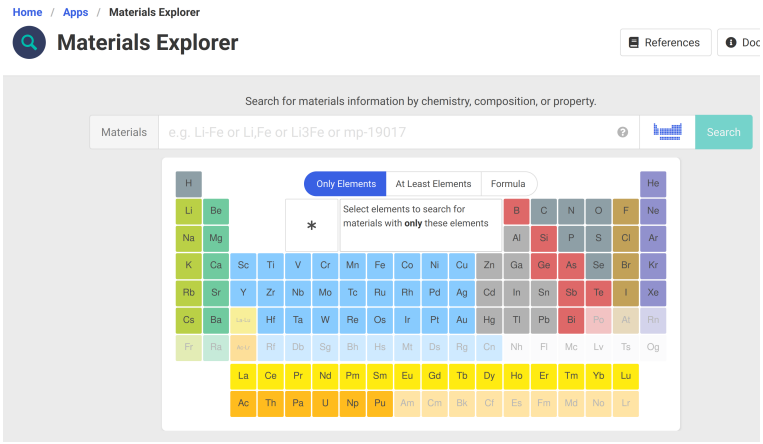
▶ Investigate the **sensitivity** of the calculated bulk modulus of diamond to the choice of η, by repeating the calculations for $\eta = 0.1$, 0.01, and 0.001. Plot the dependence of the bulk modulus on η.

Elastic constants of STO

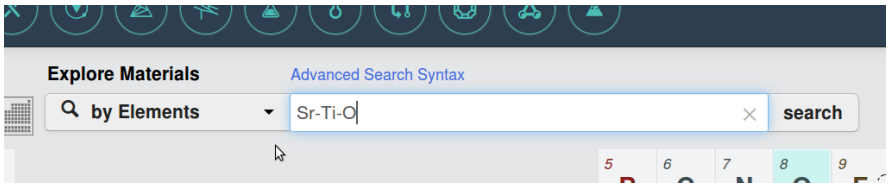
In this exercise we want to set up a simple input file to study $SrTiO_3$. In order to find an initial guess for the unit cell and atomic coordinates, we search the Materials Project database.

In order to search the Materials Project we need to create an account. To this aim, please go to `https://www.materialsproject.org` and click on Sign in or Register. The registration process only requires an email address and a password, this should take less than a minute to complete.

After logging-in you should be able to click on "Start Exploring Materials" and see a periodic table like the following:

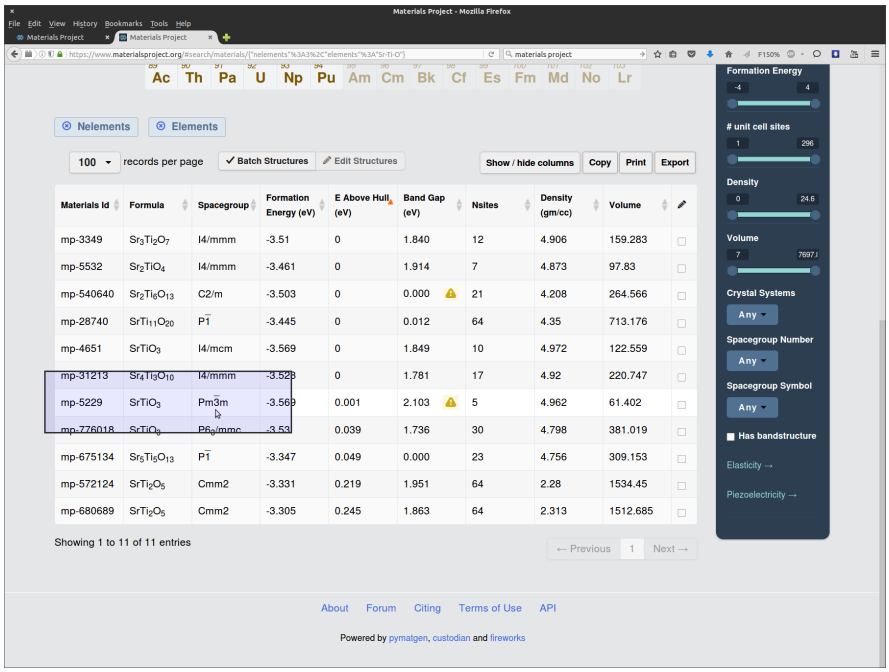


Now we can search for $SrTiO_3$ by entering Sr, Ti, and O in the search bar:

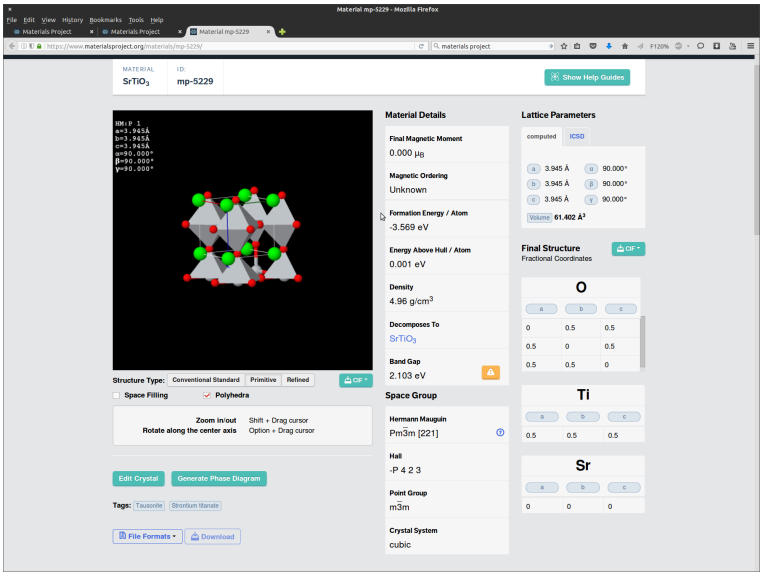


We are looking for the $Pm\bar{3}m$ structure of $SrTiO_3$, which is the high-temperature cubic phase. We will study the cubic

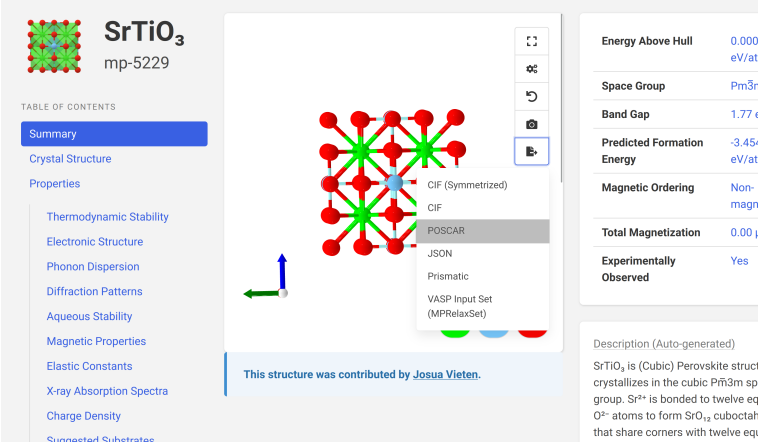
phase since it only contains 5 atoms, therefore calculations are relatively easy.



By clicking on the $Pm\bar{3}m$ field we are shown the properties of this structures that have been uploaded in the database:



All we need from this page is the structural data, which can be found in the `poscar` file:



The 'poscar' format is the standard format of VASP, another widely used sofwtare packages for DFT calculations using planewaves and pseudopotentials.

The 'poscar' file thus downloaded should look like the following:

```
1   Sr1 Ti1 O3
2   1.0
3   3.9127 0.0000 0.0000
4   0.0000 3.9127 0.0000
5   0.0000 0.0000 3.9127
6   Sr Ti O
7   1 1 3
8   direct
9   0.000000 0.000000 0.000000 Sr2+
10  0.500000 0.500000 0.500000 Ti4+
11  0.500000 0.000000 0.500000 O2-
12  0.500000 0.500000 0.000000 O2-
13  0.000000 0.500000 0.500000 O2-
```

Here the first line is a comment field, the second line contains the lattice parameter a in Å. Lines 3–5 contain the lattice vectors, scaled by the lattice parameter a: $\mathbf{a}_1/a$, $\mathbf{a}_2/a$, $\mathbf{a}_3/a$ (note that in this example the authors decided to set $a = 1$ so as to give the lattice vectors directly in Å). Line 6 contains the list of atoms in the unit cell, followed by the number of atoms of each type on line 7, in the same order. The keyword 'direct' on line 8 specifies that the atomic coordinates in lines 9-13 are expressed as fractional coordinates (units of 'direct' lattice), eg the Ti atom is at $(\mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3)/2$.

▶ Construct the input file for a total energy calculation of $SrTiO_3$ using `pw.x`. For the time being we can leave the pseudopotential field blank. You can start by modifying an input file from a previous exercise. If anything is unclear, please consult the documentation at `https://www.quantum-espresso.org/Doc/INPUT_PW.html`. Remember that `celldm(1)` is to be given in atomic units. To specify that the atomic positions are in crystal coordinates we use the keyword: `ATOMIC_POSITIONS crystal`.

At this point we need pseudopotentials for Sr, Ti, O. For this exercise we want to use the LDA exchange and correlation functional, therefore we need to identify LDA pseudopotentials. We are looking for pseudos with the label `pz` (Perdew-

Zunger) in the filename. In order to make sure that we all obtain the same results, let us decide that we consider only pseudopotentials generated by Hartwigsen, Goedeker & Hutter. These pseudos can be found here: pseudopotentials.quantum-espresso.org/legacy_tables/hartwigesen-goedecker-hutter-pp and are `Sr.pz-hgh.UPF`, `Ti.pz-hgh.UPF`, and `O.pz-hgh.UPF`.

▶ Download the required pseudopotential files from the QE library, using the `wget` command.

▶ Show your complete input file for STO in your homwework assignment, so that we can check that everything is in the right place.

▶ Using the input file just created, say `sto-1.in`, perform a test run using `calculation = 'scf'`. This test is only to make sure that everything goes smoothly. For this test we can use some arbitrary convergence parameters, say `ecutwfc = 40` and a Brillouin-zone sampling `4 4 4 1 1 1`.

Convergence tests for STO

Now that we have a basic setup for $SrTiO_3$, we need to perform convergence tests.

▶ Determine the planewaves kinetic energy cutoff that is required to have the total energy converged to within 50 meV/atom. For this calculation you can use the same `K_POINTS` settings as in the previous exercise. As a reminder, we performed this kind of tests for silicon in HandsOn 02.

▶ Using the cutoff just obtanied, determine the sampling of the Brillouin zone required to have the total energy converged to within 10 meV/atom. As a reminder, we performed this kind of tests for silicon in HandsOn 02.

Structure optimization for STO

▶ Using the convergence parameters obtained in Exercise 3, determine the optimized lattice parameter of $SrTiO_3$ by using a calculation of type `vc-relax`, as in HandsOn 5.

In this calculation you will note that the residual pressure at the end of the run is still nonzero. In order to fully optimize the lattice parameter, it is convenient to perform one or two additional runs using the optimized parameter as a starting point.

▶ Determine the optimized lattice constant of STO once again, this time by mapping the potential energy surface as a function of the lattice parameter, as in HandsOn 3.

▶ Which calculation provides the lowest total energy, the automatic optimization or the manual calculation of the potential energy surface?

▶ Compare your optimized lattice parameter with the experimental value from Cao et al, PSSA 181, 387 (2000).

Bulk modulus of STO

As a sanity check, at the end of the last two exercises you should have obtained the following parameters:

```
...
celldm(1) = 7.2266,
ecutfwc = 210,
...
K_POINTS
```

```
4 4 4 1 1 1
```

▶ Use these parameters to calculate the bulk modulus of cubic $SrTiO_3$. For this calculation, follow the same procedure as for the bulk modulus of diamond (i.e. use an isotropic deformation). Indicate the total energies and volume that you are using to calculate the bulk modulus using central finite differences.

▶ Compare your calculated bulk modulus with the experimental values of Bell & Rupprecht, Phys. Rev. 129, 90 (1963). What is the relative error between your result and experiment?

Hands-On 7
Vibrations and phonon dispersions

In this session we will learn how to calculate the vibrational frequencies of molecules and solids and phonon dispersion relations.

Stretching frequency of a diatomic molecule

We start from the simplest possible system, the diatomic molecule Cl_2 studied in HandsOn 03.

We copy the pseudopotential, executable, and input file from HandsOn 03:

```
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x ./
$ cp ~/PHY392Q/HandsOn03/cl2/Cl.pz-bhs.UPF ./
$ cp ~/PHY392Q/HandsOn03/cl2/cl2.in ./
```

For the input file we use the following, from HandsOn 03. Here we use the optimized geometry and convergence parameters.

```
&control
 calculation = "scf"
 prefix = "cl2",
 pseudo_dir = "./",
 outdir = "./"
/
&system
 ibrav = 1,
 celldm(1) = 20.0,
 nat = 2,
 ntyp = 1,
 ecutwfc = 100,
/
&electrons
/
ATOMIC_SPECIES
 Cl 1.0 Cl.pz-bhs.UPF
ATOMIC_POSITIONS bohr
 Cl 0.000 0.00 0.00
 Cl 3.725 0.00 0.00
K_POINTS gamma
```

As usual we perform a test run to make sure that everything goes smoothly:

```
$ ibrun -n 4 pw.x < cl2.in
```

In Lecture 10 we have seen that the vibrational frequency of a diatomic molecule can be calculated using:

$$\omega = \sqrt{\frac{2K}{M}}, \qquad K = \left.\frac{\partial^2 U}{\partial d^2}\right|_{d_0}$$

where M is the mass of the Cl nucleus, U is the total potential energy surface, d is the Cl-Cl distance, and d_0 is the equilibrium bond length.
By approximating the second derivative using finite differences we have:

$$\hbar\omega \simeq \hbar\sqrt{\frac{2}{M}\frac{U(d_0+\delta) - 2U(d_0) + U(d_0-\delta)}{\delta^2}}$$

where δ is a small number, say $\delta = 0.001$ bohr.

We now calculate $U(d_0)$, $U(d_0+\delta)$, and $U(d_0-\delta)$ by creating two new input files where the coordinates of the second Cl atom are modified.
We can do this as usual using `vi`. Alternatively we can use the following strategy, which is slightly faster:

```
$ sed "s/3.725/3.726/g" cl2.in > cl2_plus.in
$ sed "s/3.725/3.724/g" cl2.in > cl2_minus.in
```

Here we are replacing the distance 3.725 bohr by 3.726 and 3.724, respectively. It is convenient to extract the corresponding total energies from the output files on the fly. This can be done as follows:

```
$ ibrun -np 8 pw.x < cl2.in | grep "\!" > U0.txt
$ ibrun -np 8 pw.x < cl2_plus.in | grep "\!" > U_plus.txt
$ ibrun -np 8 pw.x < cl2_minus.in | grep "\!" > U_minus.txt
```

In these expressions the vertical bar (`|`) 'pipes' the output from the command on the left (`ibirun -np 8 pw.x < cl2.in`) into the input of the following command (`grep "\!"`). The output of `grep "\!"` is then 'redirected' (`>`) into the file on the right (`U0.txt`).

After executing these commmands we should see the following:

```
$ more U*.txt
::::::::::::::
U0.txt
::::::::::::::
!    total energy              =     -59.99059537 Ry
::::::::::::::
U_minus.txt
::::::::::::::
!    total energy              =     -59.99059533 Ry
::::::::::::::
U_plus.txt
::::::::::::::
!    total energy              =     -59.99059498 Ry
```

At this point we can combine our results, considering that the mass of Cl is 35.45 amu (1 amu = 1822.8885 m_e). We

find:

$$\hbar\omega = 69.4 \text{ meV}$$

to be compared to the experimental value of 66.7 meV.

▶ Show the steps that you followed to obtain the vibrational frequency of the Cl_2 molecule in meV, including the detail of the conversions between units.

▶ Repeat the calculation of the stretching frequency of Cl_2, this time using (i) $\delta = 0.002$ bohr, (ii) $\delta = 0.01$ bohr.

Stretching frequency of a diatomic molecule, using DFPT

The calculation method of the previous section is very general and widely used, however there exists a faster alternative based on **density-functional perturbation theory** (DFPT).

In DFPT the vibrational frequency is calculated directly by working with the equilibrium structure, using perturbation theory. The perturbation theory used in this case is similar to what you learned in introductory quantum mechanics, only slightly more complicated because the change of the self-consistent potential involves the change of the electron density.

In the Quantum ESPRESSO package, DFPT for lattice vibrations is implemented in a code named `ph.x`. In order to use this code we need to go back to the root directory `/PHY392Q/q-e-qe-7.5`, and execute:

```
$ make ph
$ cp bin/ph.x $SCRATCH
```

We can build a simple input file for Cl_2 as follows:

```
$ cat > cl2_ph.in << EOF
vibrations of cl2
&inputph
prefix = "cl2",
amass(1) = 35.45,
outdir = "./",
fildyn = "cl2.dyn",
/
0.0 0.0 0.0
EOF
```

Here the first line is just a comment field; `prefix` must be the same as that used by `pw.x`; `amass` is the atomic mass in amu (atomic mass units); `fildyn` specifies the output file that will contain the dynamical matrix. The last line specifies that we want a calculation at the Γ point, that is $\mathbf{q} = (0, 0, 0)$. This is appropriate since we are considering an isolated molecule.

In order to execute `ph.x` we first need to calculate the ground state properties of the system using `pw.x`. In this case we must modify the input file `cl2.in` as follows:

```
$ cat > cl2_pw.in << EOF
&control
 calculation = "scf"
 prefix = "cl2",
```

```
 pseudo_dir = "./",
 outdir = "./"
/
&system
 ibrav = 1,
 celldm(1) = 20.0,
 nat = 2,
 ntyp = 1,
 ecutwfc = 100,
/
&electrons
/
ATOMIC_SPECIES
 Cl 1.0 Cl.pz-bhs.UPF
ATOMIC_POSITIONS bohr
 Cl 0.000 0.00 0.00
 Cl 3.725 0.00 0.00
K_POINTS tpiba
1
0.0 0.0 0.0 1.0
EOF
```

With this modification `pw.x` is still instructed to calculate wavefunctions at Γ, that is $\mathbf{k} = 0$. The difference between this file and the version that we have used on pag. 1 of this tutorial is that now we are instructing `pw.x` to treat wavefunctions as complex quantities; in the previous version, the keyword `gamma` was instructing the code to treat wavefunctions as real quantities.
The results do not change, but this modification is needed because `ph.x` only recognizes complex wavefunctions.

We can now execute `pw.x` and `ph.x` as usual:

```
$ ibrun -np 8 pw.x < cl2_pw.in > cl2_pw.out
$ ibrun -np 8 ph.x < cl2_ph.in > cl2_ph.out
```

The phonon calculation may take a few minutes, so you might want to use 24 cores in the execution. After completion of this job we should find the file `cl2.dyn` in our working directory:

```
$ more cl2.dyn
```

```
Dynamical matrix file
vibrations of cl2
  1    2  1 20.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
           1  'Cl  '    32310.698418201875
    1    1       0.0000000000      0.0000000000      0.0000000000
    2    1       0.1862500000      0.0000000000      0.0000000000

     Dynamical  Matrix in cartesian axes

     q = (    0.000000000   0.000000000   0.000000000 )

    1    1
  0.39601498  0.00000000     0.00000000  0.00000000     0.00000000  0.00000000
```

```
 0.00000000  0.00000000     0.00242115  0.00000000     0.00000000  0.00000000
 0.00000000  0.00000000     0.00000000  0.00000000     0.00242115  0.00000000
   1    2
-0.42793567  0.00000000     0.00000000  0.00000000     0.00000000  0.00000000
 0.00000000  0.00000000     0.00470496  0.00000000     0.00000000  0.00000000
 0.00000000  0.00000000     0.00000000  0.00000000     0.00470496  0.00000000
   2    1
-0.42794215  0.00000000     0.00000000  0.00000000     0.00000000  0.00000000
 0.00000000  0.00000000     0.00470361  0.00000000     0.00000000  0.00000000
 0.00000000  0.00000000     0.00000000  0.00000000     0.00470361  0.00000000
   2    2
 0.39529479  0.00000000     0.00000000  0.00000000     0.00000000  0.00000000
 0.00000000  0.00000000     0.00156827  0.00000000     0.00000000  0.00000000
 0.00000000  0.00000000     0.00000000  0.00000000     0.00156827  0.00000000

    Diagonalizing the dynamical matrix

    q = (    0.000000000   0.000000000   0.000000000 )

 **************************************************************************
    freq (    1) =      -3.288628 [THz] =    -109.696808 [cm-1]
(  0.706777 -0.000000 -0.006470  0.000000  0.000207 -0.000000 )
(  0.707373 -0.000000  0.006931 -0.000000  0.000481 -0.000000 )
    freq (    2) =      -0.958353 [THz] =     -31.967219 [cm-1]
(  0.002775  0.000000  0.292747  0.000000 -0.606624  0.000000 )
(  0.002799  0.000000 -0.323416  0.000000  0.664600  0.000000 )
    freq (    3) =      -0.953340 [THz] =     -31.800001 [cm-1]
(  0.006126  0.000000  0.607090  0.000000  0.295678  0.000000 )
(  0.006071  0.000000 -0.664119  0.000000 -0.320762  0.000000 )
    freq (    4) =       1.498983 [THz] =      50.000706 [cm-1]
(  0.000264  0.000000  0.277596  0.000000 -0.684841  0.000000 )
(  0.000443  0.000000  0.250872  0.000000 -0.625297  0.000000 )
    freq (    5) =       1.501288 [THz] =      50.077587 [cm-1]
( -0.000293  0.000000  0.684573  0.000000  0.274908  0.000000 )
(  0.000172  0.000000  0.625591  0.000000  0.253815  0.000000 )
    freq (    6) =      16.609573 [THz] =     554.035711 [cm-1]
(  0.707404  0.000000  0.000239  0.000000 -0.000018  0.000000 )
( -0.706809  0.000000  0.000261  0.000000  0.000028  0.000000 )
 **************************************************************************
```

Here the blue lines represent the calculated dynamical matrix: we have 2 atoms and 3 Cartesian coordinates, therefore the size of this matrix is 6×6. The blue lines correspond to precisely 36 numbers, presented as pairs of real and imaginary part. The numbers in red are the vibrational frequencies obtained by diagonalizing the dynamical matrix.

The vibrational frequencies are usually given in units of cm^{-1}, i.e. as a **wavenumber**. The conversion is:

$$1 \text{ meV} = 8.0655 \text{ cm}^{-1}$$

In the dynamical matrix file we see that some frequencies are negative. This is only a **convention**, and it is meant to indicate that the diagonalization of the dynamical matrix led to a negative eigenvalue: $\omega^2 < 0$. In these cases the code prints the quantity $-\sqrt{|\omega^2|}$, and the minus sign is just a flag to warn us that something is not right. In other DFT codes you may find the imaginary unit next to these frequencies, eg $109.696808\,\mathbf{i}$.

In this example we were expecting to obtain $\omega = 0$ for 5 modes (3 translations of Cl_2 and 2 rotations), and one high-frequency stretching mode. Clearly this is not the case in the above output file. What is happening here is that our Cl_2

molecule is in a **periodic** supercell, therefore a global rotation of all the molecules must involve some small amount of energy. Furthermore, in these calculations the 3D space is not exactly 'isotropic', because we are using a finite planewaves cutoff. Together these two effect lead to nonzero frequencies in modes 1–5.

These artifacts can be corrected by imposing so-called **acoustic sum rules**. In Lecture 10 we have discussed the simplest sum rule for the monoatomic chain in one dimension. In general, one can impose sum rules to make sure that the molecule will not experience any restoring force when translated or rotated. These sum rules modify the interatomic force constant matrix. We can perform this operation by calling a **post-processing** program called `dynmat.x`:

```
$ cp ~/PHY392Q/q-e-qe-7.5/bin/dynmat.x ./
$ cat > cl2.dynmat.in << EOF
&input
 fildyn = "cl2.dyn",
 asr = "zero-dim",
/
EOF
$ ./dynmat.x < cl2.dynmat.in
```

Here `asr` is a flag that instructs the code to impose the acoustic sum rule (a.s.r.). Note that we are executing this small program in serial on the current node, without requesting many cores. This program will produce the following frequencies:

```
# mode    [cm-1]    [THz]      IR
    1      0.00    0.0000    0.0000
    2      0.00    0.0000    0.0000
    3      0.00    0.0000    0.0000
    4      0.00    0.0000    0.0000
    5      0.00    0.0000    0.0000
    6    554.04   16.6096    0.0000
```

We see that now the system has only one nonzero vibrational frequency, as expected. The calculated value 68.7 meV (1 meV = 8.0655 cm^{-1}) is close to our result from the previous section, 69.4 meV. The two values are not identical for two reasons: (1) The acoustic sum rule modifies the potential energy surface, and (2) the present calculations correspond to taking the second derivative of U in the limit $\delta \to 0$.

Note: The documentation about the phonon code `ph.x` can be found at the following link:
http://www.quantum-espresso.org/Doc/INPUT_PH.html
http://www.quantum-espresso.org/Doc/INPUT_DYNMAT.html

An extensive set of examples on how to use `ph.x` is located inside the the directory:
`~/PHY392Q/q-e-qe-7.5/PHonon/examples`

Phonon dispersion relations of diamond

In this section we calculate the phonon dispersion relations of diamond. We begin by setting up the usual input file for diamond, from HandsOn 04:

```
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x ./
$ cp ~/PHY392Q/q-e-qe-7.5/bin/ph.x ./
$ cp ~/PHY392Q/q-e-qe-7.5/bin/dynmat.x ./
$ cp ~/PHY392Q/HandsOn04/C.pz-vbc.UPF ./
$ cp ~/PHY392Q/HandsOn04/diamond_optimized.in ./diamond.in
$ more diamond.in
```

```
&control
 calculation = 'scf'
 prefix = 'diamond',
 pseudo_dir = './',
 outdir = './'
/
&system
 ibrav = 2,
 celldm(1) = 6.66407,
 nat = 2,
 ntyp = 1,
 ecutwfc = 100.0,
/
&electrons
 conv_thr = 1.0d-12
/
ATOMIC_SPECIES
 C 1.0 C.pz-vbc.UPF
ATOMIC_POSITIONS crystal
 C 0.00 0.00 0.00
 C 0.25 0.25 0.25
K_POINTS automatic
 4 4 4 1 1 1
```

Here the lattice constant, the Brillouin-zone sampling, and the planewaves cutoff are set to the same values that we obtained in HandsOn 4. We are now using a threshold for the self-consistent cycle, `conv_thr`, which is more stringent than the default value. This is important since phonon calculations are quite sensitive to the accuracy of the ground-state DFT calculation.

In order to calculate phonon frequencies along some high-symmetry paths in the Brillouin zone, we need to go through three separate steps:

1) Calculate the frequencies on a uniform grid of $\mathbf{q}$-points;
2) Calculate the real-space interatomic force constants associated with the grid of step 1;
3) Calculate the frequencies along the chosen path of $\mathbf{q}$-points, using a Fourier interpolation of the interatomic force constants of step 2.

The **first step** is performed using `ph.x`:

```
$ cat > diamond_ph.in << EOF
this is a comment line
&inputph
 prefix = "diamond",
 ldisp = .true.
 amass(1) = 12.0107,
 fildyn = "dyn",
 nq1 = 2,
 nq2 = 2,
 nq3 = 2,
 tr2_ph = 1.0d-14,
/
EOF
```

Here the flag `ldisp = .true.` specifies that we are requesting a calculation on a uniform grid. The size of this grid is specified by the variables `nq1`, `nq2`, and `nq3`. Standard grids are of the order of $4 \times 4 \times 4$ to $8 \times 8 \times 8$ points; here we use a modest $2 \times 2 \times 2$ grid only to save time.

This calculation can be performed by running `pw.x` and `ph.x` as usual:

```
$ ibrun -np 8 pw.x -npool 8 < diamond.in > diamond.out
$ ibrun -np 8 ph.x -npool 8 < diamond_ph.in > diamond_ph.out
```

The **second step** is performed using a program called `q2r.x`. This is a small post-processing program which is found in the directory `q-e-qe-7.5/bin`. The input file is very simple, and we can execute this program with a single core:

```
$ cp ~/PHY392Q/q-e-qe-7.5/bin/q2r.x ./
$ cat > diamond_q2r.in << EOF
&input
 fildyn = "dyn",
 flfrc = "diam.fc"
/
EOF
$ ./q2r.x < diamond_q2r.in
```

At the end of the execution the file `diam.fc` will contain the interatomic force constants.

For the **third step** we need a program called `matdyn.x`. This is also a small post-processing program located as usual in the Quantum ESPRESSO `/bin` folder.

```
$  cp ~/PHY392Q/q-e-qe-7.5/bin/matdyn.x ./
```

The input file is as follows:

```
$ cat > diamond_matdyn.in << EOF
&input
 asr = "simple",
 flfrc = "diam.fc",
 flfrq = "diam.freq"
```

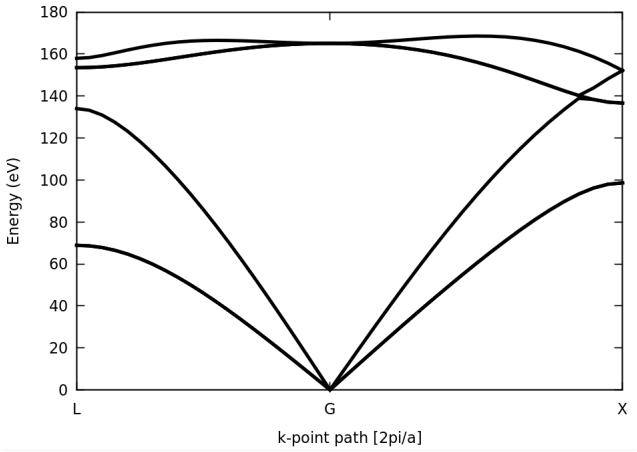
```
 q_in_band_form = .true.,
/
 3
 0.500 0.500 0.500 20
 0.000 0.000 0.000 20
 1.000 0.000 0.000 20
EOF
$ ./matdyn.x < matdyn.in
```

Here the keyword `q_in_band_form = .true.` indicates that we want the code to calculate vibrational frequencies along a path with 3 vertices in the Brillouin zone, from $(1/2, 1/2, 1/2)2\pi/a$ to $(0, 0, 0)$, and from $(0, 0, 0)$ to $(1, 0, 0)2\pi/a$. Along each segment we will have 20 **q**-points. The Cartesian coordinates of these points are specified in units of $2\pi/a$. In this example we are considering the path $L \rightarrow \Gamma \rightarrow X$. L is $(1/2, 1/2, 1/2)2\pi/a$, X is $(1, 0, 0)2\pi/a$, and Γ is $(0, 0, 0)$.

The calculated frequencies can be found in the file `diam.freq.gp`. A plot of these data using `gnuplot` can be performed as follows:

In case you are using `gnuplot`, the above was obtained as follows, after entering `gnuplot`:

```
unset key
set xlabel "k-point path [2pi/a]"
set xtics ("L" 0.0, "G" 0.866, "X" 1.866)
set ylabel "Energy (eV)"
plot [0:1.866] for [i=2:7] "diam.freq.gp" u 1:(column(i)/8.0655) w l lc 3 lw 2
```



▶ Repeat all the steps performed in this exercise for diamond for the following cases: (i) `nq1=1, nq2=1, nq3=1` in `diamond_ph.in`; (ii) `nq1=2, nq2=2, nq3=2` in `diamond_ph.in`; (iii) `nq1=4, nq2=4, nq3=4` in `diamond_ph.in`.

▶ Compare the phonon dispersion relations that you obtained in (i), (ii), (iii) by overlaying the curves in the same plot. What can you deduce from this comparison?

Hands-On 8
Phonons in polar crystals and LO-TO splitting

In this session we study the phonon dispersion relations of two polar crystals, **GaAs** and **$SrTiO_3$**.

LO-TO splitting, IR activity, and dielectric constant of GaAs

We consider GaAs as an example of **polar** semiconductor. The atoms of polar semiconductors exhibit nonzero Born effective charges. The main consequences of nonzero Born charges are:

1. The vibrational frequencies of longitudinal and transverse optical phonons at long wavelength ($\mathbf{q} \to 0$) do not coincide. In the theory part we said that this efeffect is called LO-TO splitting.

2. The system exhibits infrared (IR) activity.

3. The ionic vibrations provide an additional contribution to the dielectric constant at low frequency.

Let us create a basic input file for `pw.x`, for the case of GaAs:

```
$ cd $SCRATCH
$ wget http://pseudopotentials.quantum-espresso.org/upf_files/Ga.pz-bhs.UPF
$ wget http://pseudopotentials.quantum-espresso.org/upf_files/As.pz-bhs.UPF
$ cat > GaAs_scf.in << EOF
&control
 calculation = "scf"
 prefix = "gaas",
 pseudo_dir = "./",
 outdir = "./"
/
&system
 ibrav = 2,
 celldm(1) = 10.47494,
 nat = 2,
 ntyp = 2,
 ecutwfc = 40.0,
/
&electrons
/
ATOMIC_SPECIES
 Ga 1.0 Ga.pz-bhs.UPF
 As 1.0 As.pz-bhs.UPF
ATOMIC_POSITIONS crystal
 Ga 0.00 0.00 0.00
 As 0.25 0.25 0.25
K_POINTS automatic
6 6 6 1 1 1
EOF
```

► Perform convergence tests for the planweaves cutoff and the Brillouin zone sampling, and show that the energy is converged to within 5 meV/atom error at `ecutwfc = 40` Ry, and within 0.1 meV/atom at `6 6 6 1 1 1`.

We perform a test run to make sure that everything is in place: as usual we call `pw.x`:

```
$ ibrun -np 4 pw.x < GaAs_scf.in
```

Now we calculate vibrational frequencies at $\mathbf{q} = 0$. The input file for `ph.x` is similar to what we have seen in the previous session with diamond. The only differences are the two additional flags `epsil` and `zeu`:

```
$ cat > GaAs_ph.in << EOF
phonons of GaAs
&inputph
 prefix = "gaas",
 amass(1) = 69.72,
 amass(2) = 74.92,
 epsil = .true.,
 zeu = .true.,
 fildyn = "dyn",
 tr2_ph = 1.0d-14,
/
0.0 0.0 0.0
EOF
```

If we visit the documentation page, http://www.quantum-espresso.org/Doc/INPUT_PH.html, and look for these flags we find:

epsil	LOGICAL
Default:	.false.

If .true. in a q=0 calculation for a non metal the macroscopic dielectric constant of the system is computed. Do not set epsil to .true. if you have a metallic system or q/=0: the code will complain and stop.

zeu	LOGICAL
Default:	zeu=epsil

If .true. in a q=0 calculation for a non metal the effective charges are computed from the dielectric response. This is the default algorithm. If epsil=.true. and zeu=.false. only the dielectric tensor is calculated.

Therefore these flags instruct `ph.x` to also evaluate the high-frequency (electronic) dielectric constant tensor $\varepsilon^{\infty}_{\alpha\beta}$ of the system, as well as the Born effective charges $Z^{*}_{\kappa,\alpha\beta}$. As we have seen in in the theory lecture, these quantities are needed for calculating the IR activity of each mode and the static dielectric constant.

We execute `ph.x` using this input file from our batch script:

```
$ ibrun -n 8 ph.x -npool 8 < GaAs_ph.in > GaAs_ph.out
```

Near the end of the output file we find the following information:

```
     Number of q in the star =    1
     List of q in the star:
```

```
          1   0.000000000   0.000000000   0.000000000

          Dielectric constant in cartesian axis

          (      11.565843995      -0.000000000      -0.000000000 )
          (      -0.000000000      11.565843995       0.000000000 )
          (      -0.000000000       0.000000000      11.565843995 )

          Effective charges (d Force / dE) in cartesian axis with asr applied:

               atom      1   Ga Mean Z*:        2.03631
          E*x (        2.03631       -0.00000       -0.00000 )
          E*y (       -0.00000        2.03631        0.00000 )
          E*z (       -0.00000        0.00000        2.03631 )
               atom      2   As Mean Z*:       -2.03631
          E*x (       -2.03631        0.00000        0.00000 )
          E*y (        0.00000       -2.03631       -0.00000 )
          E*z (        0.00000       -0.00000       -2.03631 )

     Diagonalizing the dynamical matrix

     q = (    0.000000000   0.000000000   0.000000000 )

 **************************************************************************
     freq (    1) =      -0.231896 [THz] =      -7.735217 [cm-1]
     freq (    2) =      -0.231896 [THz] =      -7.735217 [cm-1]
     freq (    3) =      -0.231896 [THz] =      -7.735217 [cm-1]
     freq (    4) =       8.262471 [THz] =     275.606365 [cm-1]
     freq (    5) =       8.262471 [THz] =     275.606365 [cm-1]
     freq (    6) =       8.262471 [THz] =     275.606365 [cm-1]
 **************************************************************************
```

Here we recognize the high-frequency dielectric constant of GaAs, $\epsilon_\infty = 11.57$, and the Born effective charges of Ga ans As, respectively $Z^*_{\rm Ga} = 2.04$ and $Z^*_{\rm As} = -2.04$.

The calculated dielectric constant is about 6% higher than the experimental value, $\epsilon_\infty^{\rm exp} = 10.89$. This overestimation is related to the band gap problem of DFT, which will be discussed in Lecture 11.

In the above output of `ph.x` we can see that the three translational modes at $\mathbf{q} = 0$ have an imaginary/negative frequency. This is once again a numerical artifact, and can be corrected by trying other acousting sum rules.

In the same output file we see that the three optical modes exhibit three identical frequencies, while we were expecting two degenerate TO modes and one LO mode at a higher frequency. The reason why the three optical modes are degenerate is that the calculation performed by `ph.x` missed a contribution, called the non-analytical part of the dynamical matrix. This contribution can be calculated by using the dielectric constant and Born charges. Let us see how:

We create a new input file for `dynmat.x`:

```
$ cat > GaAs_dynmat.in << EOF
&input
 fildyn = "dyn",
 asr = "simple",
 lperm = .true.,
 q(1)=1.0,
```

```
 q(2)=0.0,
 q(3)=0.0
/
EOF
```

Here `q(1)`, `q(2)`, and `q(3)` specify the direction along which we approach $\mathbf{q} \to 0$ (the LO-TO splitting is direction-dependent). When one of these numbers is nonzero, `dynmat.x` understands that it must read the dielectric constant and Born charges, calculate the LO-TO correction, and determine IR activities. The additional flag `lperm` specifies that we also want the static dielectric permittivity. After running `dynmat.x` we should obtain the following:

```
$ ./dynmat.x < dynmat.in

...
     IR activities are in (D/A)^2/amu units

# mode   [cm-1]    [THz]      IR
    1      0.00    0.0000    0.0000
    2      0.00    0.0000    0.0000
    3      0.00    0.0000    0.0000
    4    275.70    8.2652    2.6490
    5    275.70    8.2652    2.6490
    6    295.77    8.8671    2.6490

Electronic dielectric permittivity tensor (relative, adimensional)
        11.565844    0.000000    0.000000
         0.000000   11.565844    0.000000
         0.000000    0.000000   11.565844

 ... with zone-center polar mode contributions
        13.082633    0.000000    0.000000
         0.000000   13.311575    0.000000
         0.000000    0.000000   13.311575
```

We can see that now we have 2 degenerate TO modes at 34.18 meV, and 1 LO mode at 36.67 meV. The corresponding LO-TO splitting is 2.5 meV, in good agreement with the experimental value of 2.72 meV obtained by Strauch & Dorner, J. Phys. Condens. Matter 2, 1457 (1990).

The dielectric constants are highlighted in magenta. Here we see that $\epsilon_0 = 13.2$ (isotropic average: the values along the diagonal are slightly different due to numerical errors). For comparison the experimental value is $\epsilon_0^{\rm exp} = 12.9$, therefore the relative deviation is of only 2.3%.

In the output file we also see the IR activities in units of debye/Å^2/amu (highlighted in red above). This means that, if we measure the IR absorption spectrum of GaAs, we expect to see an absorption line at the TO frequecny of 34.18 meV. Generally IT measurements on thick films only detect TO modes; LO modes are seen in thin films at oblique incidence (Berreman effect), and the relative intensities of LO and TO modes depend on the angle of incidence and film thickness. Some illustrative measurements on III-V semiconductors can be found in Ibáñez et al, J. Appl. Phys. 104, 033544 (2008).

Note that all these calculations refer to zone-center phonons, $\mathbf{q} \to 0$.

▶ Repeat all the calculations, and show your results by copy/pasting snippets of the output files. For these calculations use a planewaves cutoff of 60 Ry.

Note. The procedure that we used for calculating the LO-TO splitting can be bypassed by performing a direct calculation of phonon frequencies using a small but nonzero wavevector. For example we could use:

```
$ cat > GaAs_ph_2.in << EOF
phonons of GaAs near Gamma
&inputph
 prefix = "gaas",
 amass(1) = 69.72,
 amass(2) = 74.92,
 fildyn = "dyn",
 tr2_ph = 1.0d-14,
/
0.01 0.0 0.0
EOF
```

This gives two degenerate TO phonons at 34.16 meV and one LO phonon at 36.66 meV, in agreement with our previous calculation.
This alternative procedure is perfectly legitimate, but (i) it does not provide us with Born charges, dielectric constants, and IR activities, and (ii) the calculation takes much longer.

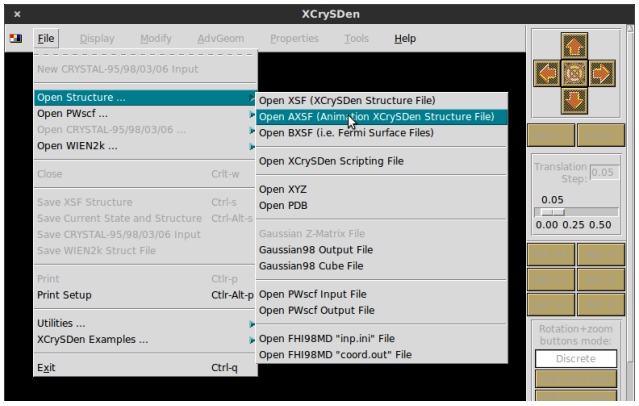
▶ Try this alternative approach and show your calculated frequencies.

Visualization of vibrational modes

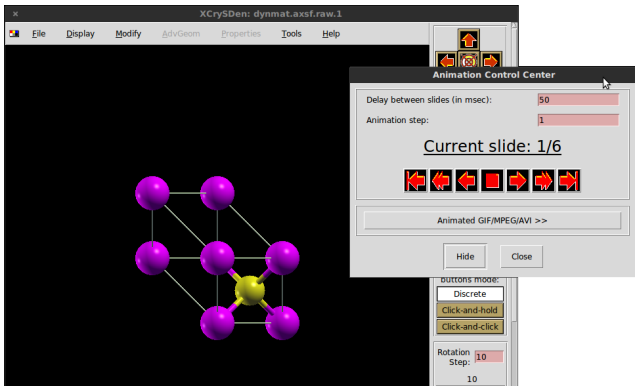
If you succeeded to complete the calculation in the previous page (`GaAs_ph.in` and `GaAs_dynmat.in`), you should now see in your working directory the file `dynmat.axsf`. This file has been produced by the code `dynmat.x` that was invoked at the very end of the exercise.

The file `dynmat.axsf` contains the vibrational eignemodes in a format which can be read and visualized by `XcrysDen`. Let us recall that these modes correspond to the atomic displacement patterns associated with a wavevector $\mathbf{q} \to 0$ along the direction x (this was specified in the input file `GaAs_dynmat.in`).

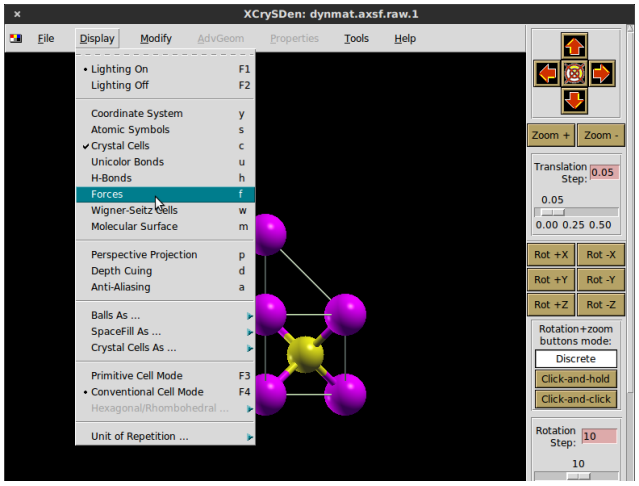
Let us call `XcrysDen` and go through the following steps. We open the `dynmat.axsf` file:



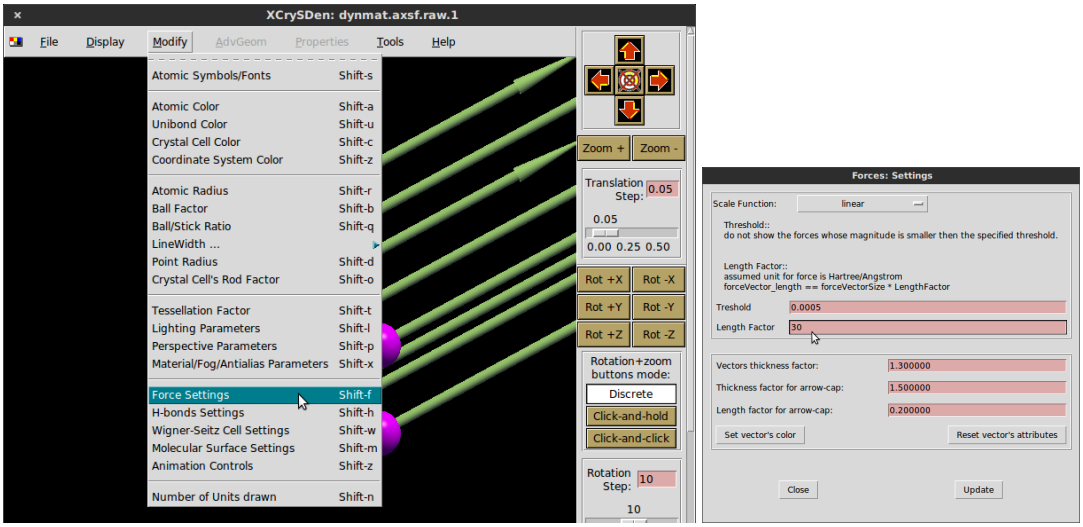
For the time being we ignore the window indicating the 'current slide'. This window will be used later to select the vibrational mode to be visualized.



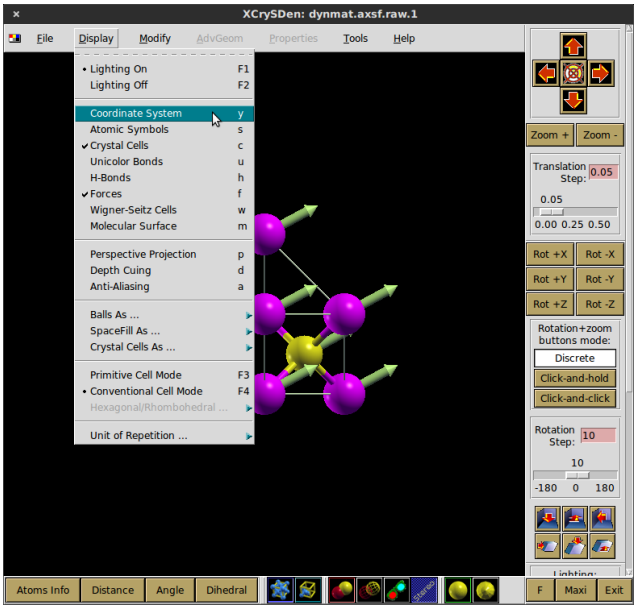
We activate the visualization of 'forces' (in our case the arrows will represent displacements, not forces, but the file format and the naming conventions are the same).



We adjust the length of the arrows by a uniform scaling:



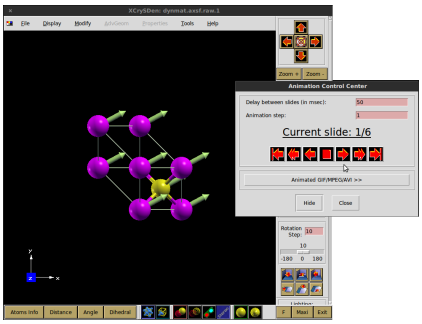
We ask the program to show the Cartesian axes:



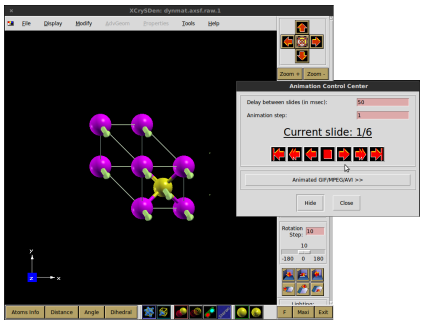
At this point we can use the window entitled ‘Current Slide’ to inspect each vibrational modes.

The result should look similar to the following:

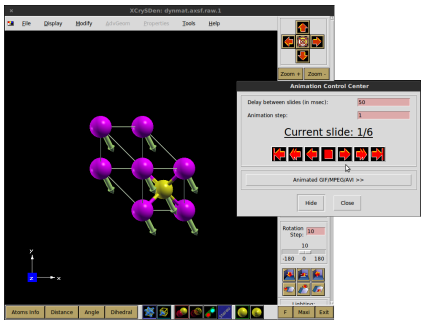
mode 1



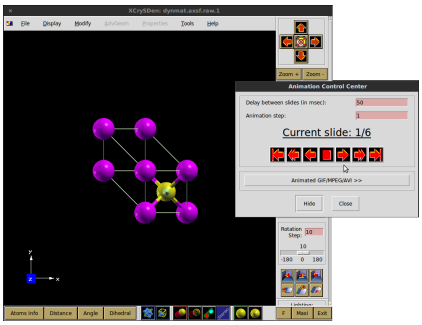
mode 2



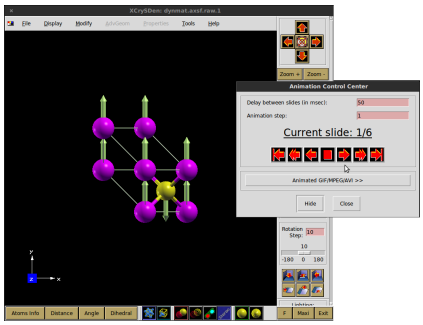
mode 3



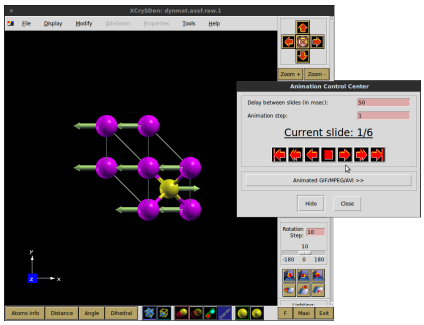
mode 4



mode 5



mode 6



Here we recognize the three translational modes at $\mathbf{q} = 0$ (modes 1–3), which should have $\omega = 0$. We can also recognize the optical phonons (modes 4–6): in these modes Ga and As atoms move in opposite directions. Furthermore, we see that in modes 4 and 5 the atomic displacements are along y and z, while in mode 6 the atoms displace along x. Since we have $\mathbf{q}$ along the x axis (see `GaAs_dynmat.in`), we can conclude that mode 6 is LO, while modes 4–5 are TO.

▶ Starting from the following `dynmat.in` input file:

```
&input
 fildyn = "dyn",
 asr = "simple",
 lperm = .true.,
 q(1)=0.0,
 q(2)=0.0,
 q(3)=1.0
/
```

Visualize the vibrational modes as described above, and identify the translational modes, the two TO modes, and the LO mode.

Phonon dispersions of GaAs including LO-TO splitting

In this exercise we need to combine what we have learned when we calculated the phonon dispersion relations of diamond, and what we did for calculating the LO-TO splitting in GaAs. The complete input file for `ph.x` is:

```
$ cat > GaAs_disp_ph.in << EOF
phonons of GaAs
&inputph
 prefix = "gaas",
 amass(1) = 69.723,
 amass(2) = 74.9216,
 epsil = .true.,
 zeu = .true.,
 fildyn = "dyn",
 tr2_ph = 1.0d-14,
 ldisp = .true.,
 nq1 = 4,
 nq2 = 4,
 nq3 = 4,
/
EOF
```

In this input file the flags in blue instruct `ph.x` to calculate the electronic dielectric permittivity tensor and the Born effective charges. These quantities are needed in order to correctly describe the LO-TO splitting. The lines in red instruct the code to calculate phonons on a uniform grid of $4 \times 4 \times 4$ **q**-points.

Note: This calculation will be time-consuming, it is recommended to use `SBATCH` and 24 cores to complete this exercise.

In order to obtain the dispersion relations, you will need to call `pw.x`, `ph.x`, `q2r.x`, and `matdyn.x` as we already did for diamond:

```
$ ibrun -n 8 pw.x -npool 4 < GaAs_scf.in > GaAs_scf.out
$ ibrun -n 8 ph.x -npool 4 < GaAs_disp_ph.in > GaAs_disp_ph.out
$ ./q2r.x < GaAs_q2r.in
$ ./matdyn.x < GaAs_matdyn.in
```

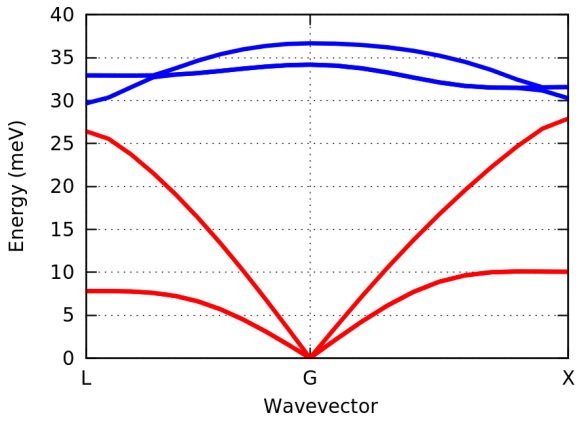
The input files `GaAs_q2r.in` and `GaAs_matdyn.in` must be prepared in the same way as for diamond in HandsOn 7. The input file `GaAs_scf.in` for GaAs is the same as the one that we used at the beginning of this Hands On session.

▶ Plot the phonon dispersion relations of GaAs along the Brillouin zone path $L\Gamma X$ (this path is identical to what we used for diamond: GaAs and diamond have the same Brilloin zone apart from the difference in lattice parameter).

▶ Verify that the LO-TO splitting at Γ is the same as that calculated at the beginning of this Hands On session.

▶ Compare your phonon dispersion relations with the inelastic neutron scattering data of Strauch & Dorner, J. Phys. Condens. Matter 2, 1457 (1990).

As a sanity check, you should be able to obtain dispersion relations resembling the following:



Phonon dispersions of $SrTiO_3$ and soft phonons

In this exercise we want to calculate the phonon dispersion relations of cubic $SrTiO_3$, including the LO-TO splitting.

▶ We will use the input file for the self-consistent calculation performed in HandsOn 6, as well as the pseudopotentials from the same exercise. Perform a test run using this setup, in order to make sure that everything goes smoothly.

▶ We now adapt the input file `ph.in` prepared in Exercise 4 for GaAs to describe $SrTiO_3$:

```
$ cat > sto_disp_ph.in << EOF
 phonons of STO
 &inputph
 prefix = "sto",
 amass(1) = 87.62,
 amass(2) = 47.867,
 amass(3) = 15.9994,
 epsil = .true.,
 zeu = .true.,
 fildyn = "dyn",
 tr2_ph = 1.0d-14,
 ldisp = .true.,
 nq1 = 2,
 nq2 = 2,
 nq3 = 2,
/
EOF
```

Now you can execute `ph.x` using this input file. You should find out that the execution takes much longer than in the case of GaAs.

Calculations on $SrTiO_3$ are more time-consuming than for GaAs since we now have 24 electrons per unit cell, and we are using a cutoff of 210 Ry. This means that we have to work with 12 Kohn-Sham wavefunctions per each $\mathbf{k}$-point, and each wavefunction is expanded in a basis of 20,000+ planewaves. In these cases it is convenient to perform calculations in two steps:

Step 1: We reduce massively the kinetic energy cutoff and the Brillouin zone sampling, and carry out the calculations until we manage to obtain our phonon dispersion relations. These results will be inaccurate and unreliable, but they will allow us to test every step very quickly.

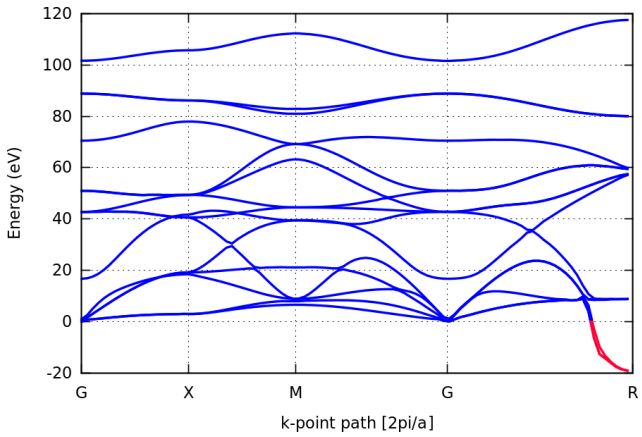
Step 2: Once we are confident about the complete procedure, we run a production calculation with convergence parameters fully optimized. This may take up to 30 min on 12 cores, but now we can be confident that the calculation will complete successfully.

▶ Following this two-step procedure, reduce the cutoff to 50 Ry and the Brillouin zone sampling to `2 2 2 1 1 1`, and calculate the phonon dispersion relations of $SrTiO_3$. The procedure is identical to what was done for GaAs on page 77.

In this case we can plot the dispersions along the high-symmetry path $\Gamma X M \Gamma R$. The Cartesian coordinates of these points are $\Gamma : (0,0,0)$, $X : (0.5,0,0)$, $M : (0.5,0.5,0)$, $R : (0.5,0.5,0.5)$ in units of $2\pi/a$. Therefore the `sto_matdyn.in` file will be:

```
$ cat > sto_matdyn.in << EOF
&input
 asr = "simple",
 flfrc = "sto.fc",
 flfrq = "sto.freq",
 q_in_band_form = .true.,
/
5
 0.000 0.000 0.000 20
 0.500 0.000 0.000 20
 0.500 0.500 0.000 20
 0.000 0.000 0.000 20
 0.500 0.500 0.500 20
EOF
```

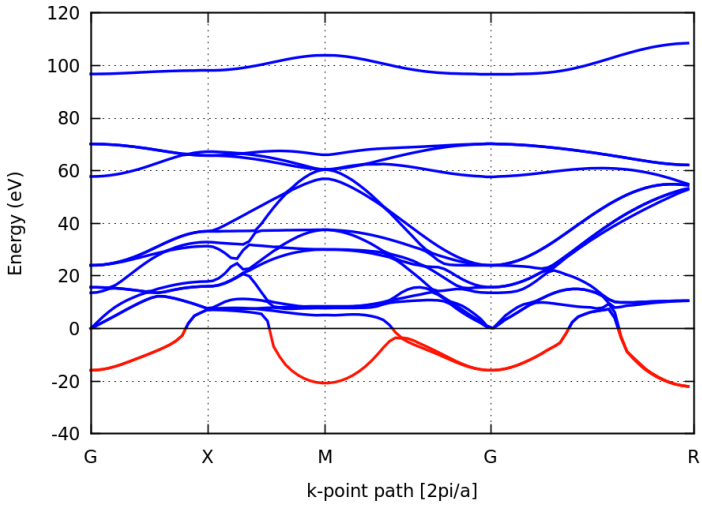
You should obtain something like the following (the imaginary frequencies are plotted in red):



▶ Now that are are confident that we can successfully complete the entire procedure, we move on to perform a production run and obtain the final dispersion relations. You can repeat the entire procedure by using the optimized parameters `ecutwfc = 210` and `K_POINTS automatic 4 4 4 1 1 1`.

For this calculation it is recommended to use 24 cores.

The final result should look as follows:



▶ Compare your result with those reported by Ghosez et al, AIP Conf. Proc. 535, 102 (2000) and by Cancellieri et al, Nat. Commun. 7, 10386 (2016).

In the last two plots, the curves in red denote imaginary frequencies, that is modes for which $\omega^2 < 0$. Since ω^2 represents the curvature of the potential energy surface, negative values imply that the system is in a local **maximum** of the energy surface, and is therefore **unstable** with respect to those vibrational modes.

▶ In order to better understand the origin of these instabilities, use `XcrysDen` to visualize the displacement patterns of the soft modes at Γ, M, and R.

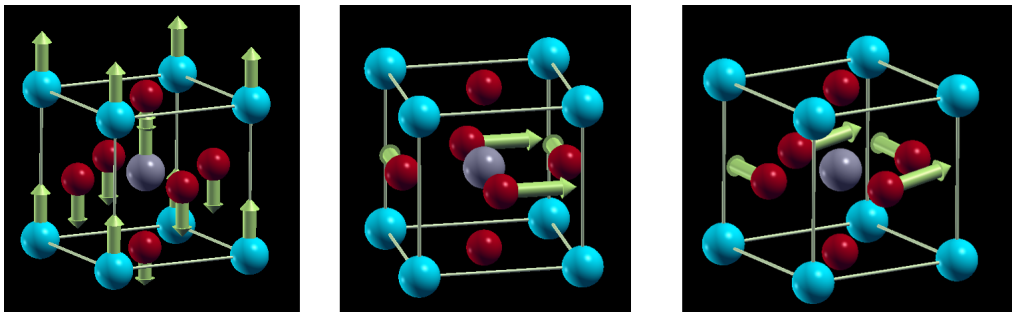
In order to use XcrysDen you will need to generate the dynmat.axsf files using dynmat.x. The dynmat.in files should look as follows:

```
$ cat > dynmat.in << EOF
&input
 fildyn = "dyn1"
/
EOF
```

where dyn1 is the file corresponding to the Γ point, dyn3 is for the M point, and dyn4 is for the R point.

The soft modes, that is the modes with imaginary frequency, should look as follows:

Γ (modes 1–3) M (mode 1) R (modes 1–3)



The soft modes at Γ indicate the presence of a ferroelectric (to be precise: quantum paraelectric) instability, the soft modes at M and R indicate instabilities against rotations of the TiO_6 octahedra.

Note: The displacements provided by dynmat.x correspond to the Bloch-periodic part of the vibrational eigenmodes. These displacements correspond to the real atomic displacements only when $\mathbf{q} = \Gamma$. In order to see the complete eigendisplacements for $\mathbf{q} \neq 0$ we would need to add the Bloch wave $\exp(i\mathbf{q} \cdot \mathbf{r})$ manually.

The origin of these soft modes lies in the fact that we are performing ground-state calculations (i.e. at 0 K) for the cubic structure of $SrTiO_3$, but the cubic structure is only stable above 110 K. These soft modes can be taken as an indication of the tendency of the system to lower its symmetry. The soft modes disappear when performing calculations on larger orthorhombic unit cells (containing 20 atoms per cell).

▶ Calculate the dielectric permittivity and IR activities of cubic $SrTiO_3$, using dynmat.x.

In this case the appropriate input file is

```
$ cat > dynmat_sto_eps.in << EOF
&input
 fildyn = "dyn1",
 asr = "simple",
 lperm = .true.,
 q(1)=1.0,
 q(2)=0.0,
 q(3)=0.0
/
EOF
```

Note that we are instructing `dynmat.x` to read data from `dyn1`, which corresponds to the Γ point.

Here we should be careful in interpreting our data: the static permittivity ϵ_0 is **not reliable** since we have soft modes with a large IR activity. Generally speaking, calculations for the high-temperature cubic phase of $SrTiO_3$ necessitate including temperature and quantum zero-point effects.

Hands-On 9
Band structures and Fermi surfaces

In this HandsOn session we will learn how to calculate the band structures of semiconductors and metals, as well as the Fermi surface of a metal.

Band structures

We start from the band structure of **silicon**. First we copy the setup from HandsOn 2:

```
$ cd $SCRATCH
$ cp ~/PHY392Q/q-e-qe-7.5/bin/pw.x  ./
$ wget http://pseudopotentials.quantum-espresso.org/upf_files/Si.pz-vbc.UPF
$ cat > si_scf.in << EOF
&control
 calculation = "scf"
 prefix = "silicon"
 pseudo_dir = "./"
 outdir = "./"
/
&system
 ibrav = 2
 celldm(1) = 10.2078
 nat = 2
 ntyp = 1
 ecutwfc = 25.0
/
&electrons
/
ATOMIC_SPECIES
 Si  28.086  Si.pz-vbc.UPF
ATOMIC_POSITIONS
 Si 0.00 0.00 0.00
 Si 0.25 0.25 0.25
K_POINTS automatic
 4 4 4 1 1 1
EOF
```

We test that everything is in place by performing the usual test run:

```
ibrun -n 4 pw.x < si_scf.in
```

Now we want to calculate the band structure. This calculation is **non self-consistent**, in the sense that we **"freeze"** the ground-state electron density, Hartree, and exchange and correlation potentials determined in the previous run. In a non self-consistent calculation, the code `pw.x` reads the ground-state electron density determined by the previous run from the file `silicon.save/charge-density.dat`. This density is used to construct the Kohn-Sham potential in the ground state. Then the Kohn-Sham Hamiltonian thus obtained is used to calculate Kohn-Sham eigenfunctions

and eigenvalues without upgrading the Kohn-Sham Hamiltonian at every step. This is achieved by using the keyword `calculation = 'bands'` and by specifying the **k**-points for which we want the eigenvalues:

```
$ cat > si_nscf.in << EOF
&control
 calculation = "bands"
 prefix = "silicon"
 pseudo_dir = "./"
 outdir = "./"
/
&system
 ibrav = 2,
 celldm(1) = 10.2078,
 nat = 2
 ntyp = 1
 ecutwfc = 25.0,
 nbnd = 8
/
&electrons
/
ATOMIC_SPECIES
 Si  28.086  Si.pz-vbc.UPF
ATOMIC_POSITIONS
 Si 0.00 0.00 0.00
 Si 0.25 0.25 0.25
K_POINTS tpiba_b
3
0.500 0.500 0.500 20
0.000 0.000 0.000 20
1.000 0.000 0.000 20
EOF
```

In this input file we are using the same path in the Brillouin zone that we used for the phonon dispersion relations of diamond in HandsOn 7. The keyword `tpiba_b` after `K_POINTS` specifies that we want `pw.x` to generate a path going through the points specified in the list. The following number (3) is the number of vertices, and the integer following the coordinates (20) is the number of points in each segment. So in this case we will have 20 points from $L = (1/2, 1/2, 1/2)2\pi/a$ to $\Gamma = (0, 0, 0)$ and 20 points from $\Gamma = (0, 0, 0)$ to $X = (1, 0, 0)2\pi/a$. The points are given in Cartesian coordinates and in units of $2\pi/a$. In this input file we also specify the number of bands that we want to calculate: in order to see the 4 valence bands of silicon and the 4 lowest conduction bands we are setting `nbnd = 8`.

After executing `pw.x`:

```
ibrun -n 4 pw.x < si_nscf.in > si_nscf.out
```

we can find the Kohn-Sham eigenvalues in the output file `si_nscf.out`
(`vi si_nscf.out` and / band):

```
...
    End of band structure calculation
```

```
          k = 0.5000 0.5000 0.5000 (   568 PWs)   bands (ev):

    -3.4441  -0.8411   4.9961   4.9961   7.7681   9.5463   9.5463  13.8005

          k = 0.4750 0.4750 0.4750 (   568 PWs)   bands (ev):

    -3.4813  -0.7883   5.0006   5.0006   7.7732   9.5517   9.5517  13.8089
...
```

Here, for each **k**-point in the input file, we have the coordinates of the point (blue) and the calculated eigenvalues in eV (red). We see 8 eigenvalues because we have requested 8 bands. Unless specified otherwise, `pw.x` assumes that each eigenvalue corresponds to 2 electrons (one with spin $\uparrow$ and one $\downarrow$). In this case the four lowest eigenvalues define the **valence bands**, and the four highest eigenvalues define **conduction bands**.

In order to plot the bands along the chosen path, we must extract these eigenvalues, and calculate the distance covered as we move along the path $L \to \Gamma \to X$. This procedure can be performed manually, for example using a shell script, or by using a small post-processing program of Quantum ESPRESSO called `bands.x`.

To use `bands.x` we must first compile it:

```
$ cd ~/PHY392Q/q-e-qe-7.5; make pp
$ cd $SCRATCH; cp  ~/PHY392Q/q-e-qe-7.5/bin/bands.x ./
```

The input file for this post-processing program is very simple: we only need to specify the prefix of the previous calculations, so that the code knows where to read data from:

```
$ cat si_bands.in << EOF
&bands
  prefix  = "silicon"
/
EOF
```

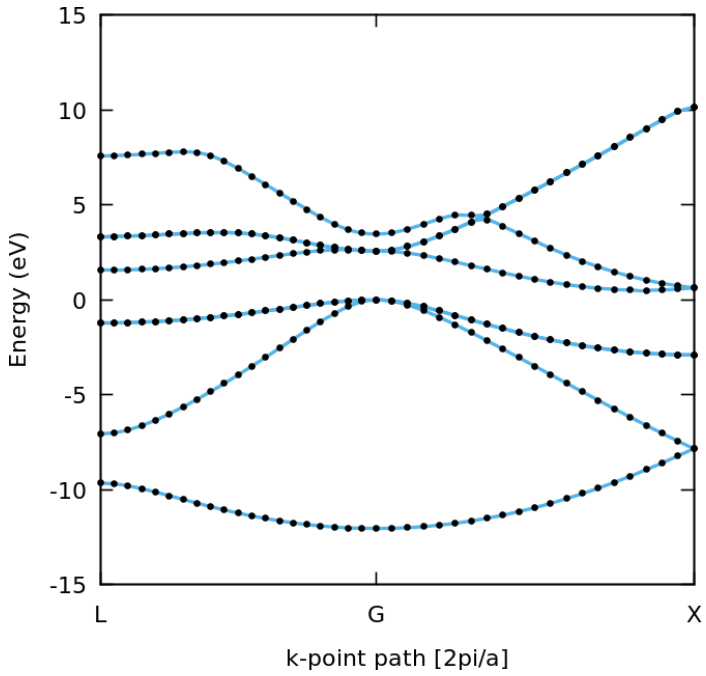
We execute this program by issuing:

```
$ ibrun -n 4 bands.x < si_bands.in
```

At this point the file `bands.gnu` contains the band structure data ordered by **k**-point, in plain text format. This file can be used to plot the bands using `gnuplot` for example:

```
ezero = 6.211
unset key
set ylabel "Energy (eV)"
set xtics ("L" 0, "G" 0.866, "X" 1.866)
set xlabel "k-point path [2pi/a]"
plot "bands.out.gnu" using 1:($2-ezero) w l lc 3 lw 2 smooth csplines, "" using 1:($2-ezero) w p pt 7 lc 0 ps 0.5
```

The result should look as follows:



Here we set the top of the valence band at Γ to zero (`ezero = 6.211`), so below 0 we have valence bands, above 0 we have conduction bands.

By looking for the valence band top at Γ and the conduction band bottom along the Γ-X line, we find that the band gap of silicon in DFT/LDA is $E_g = 0.501$ eV. The calculated band gap is much smaller than the experimental value of 1.2 eV. This is a prototypical manifestation of the **band gap problem** of DFT.

▶ Plot the band structure of silicon.

More details about the use of the program `bands.x` can be found at
https://www.quantum-espresso.org/Doc/INPUT_BANDS.html

Visualizing Kohn-Sham wavefunctions

Following the calculation of the band structure of silicon, we can visualize the wavefunctions corresponding to selected Kohn-Sham eigenvalues.

In order to plot a wavefunction we must use a post-processing code named `pp.x`. This program was already compiled when we issed `make pp` to obtain `bands.x`. We take it from the bin folder of Quantum ESPRESSO:

```
$ cp ~/PHY392T/q-e-qe-7.5/bin/pp.x ./
```

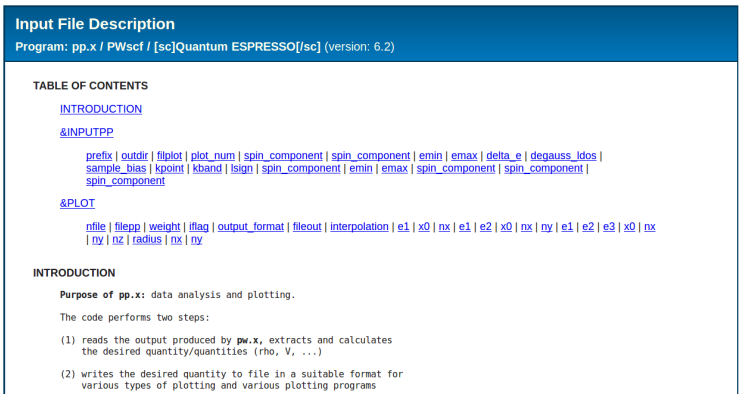
This small post-processing code reads the output of a `pw.x` run, and rewrites it in a format compatible with standard visualization software. The structure of the input file of `pp.x` is:

```
$ cat > si_pp.in << EOF
&inputpp
  prefix = "silicon"
  plot_num = 7
  lsign = .true.
  kpoint = 21
  kband = 4
/
&plot
  iflag = 3
  output_format = 5
  fileout = "silicon.xsf"
/
EOF
```

The important input variables are shown in color. `plot_num = 7` specifies that we want to plot the square modulus of Kohn-Sham wavefunctions, and the flag `lsign = .true.` is to keep track of the sign of the wavefunction. The variables `kpoint` and `kband` indicate the $\mathbf{k}$-point and band that we want to plot. In this case we are choosing the 21-st point from the list in `si_nscf.out` and the band number 4. This is precisely the valence band top at $\mathbf{k} = 0$, i.e. at Γ. The flags `iflag = 3` and `output_format = 5` specify that we want a 3D plot and that this must be in `xcrysden` format, respectively.

There are many other options for plotting other quantities of interest, for the complete range please we can consult the following documentation page:

http://www.quantum-espresso.org/Doc/INPUT_PP.html



Input File Description

Program: pp.x / PWscf / [sc]Quantum ESPRESSO[/sc] (version: 6.2)

TABLE OF CONTENTS

INTRODUCTION

&INPUTPP

prefix | outdir | filplot | plot_num | spin_component | spin_component | emin | emax | delta_e | degauss_ldos | sample_bias | kpoint | kband | lsign | spin_component | emin | emax | spin_component | spin_component | spin_component

&PLOT

nfile | filepp | weight | iflag | output_format | fileout | interpolation | e1 | x0 | nx | e1 | e2 | x0 | nx | ny | e1 | e2 | e3 | x0 | nx | ny | nz | radius | nx | ny

INTRODUCTION

Purpose of pp.x: data analysis and plotting.

The code performs two steps:

(1) reads the output produced by **pw.x**, extracts and calculates the desired quantity/quantities (rho, V, ...)

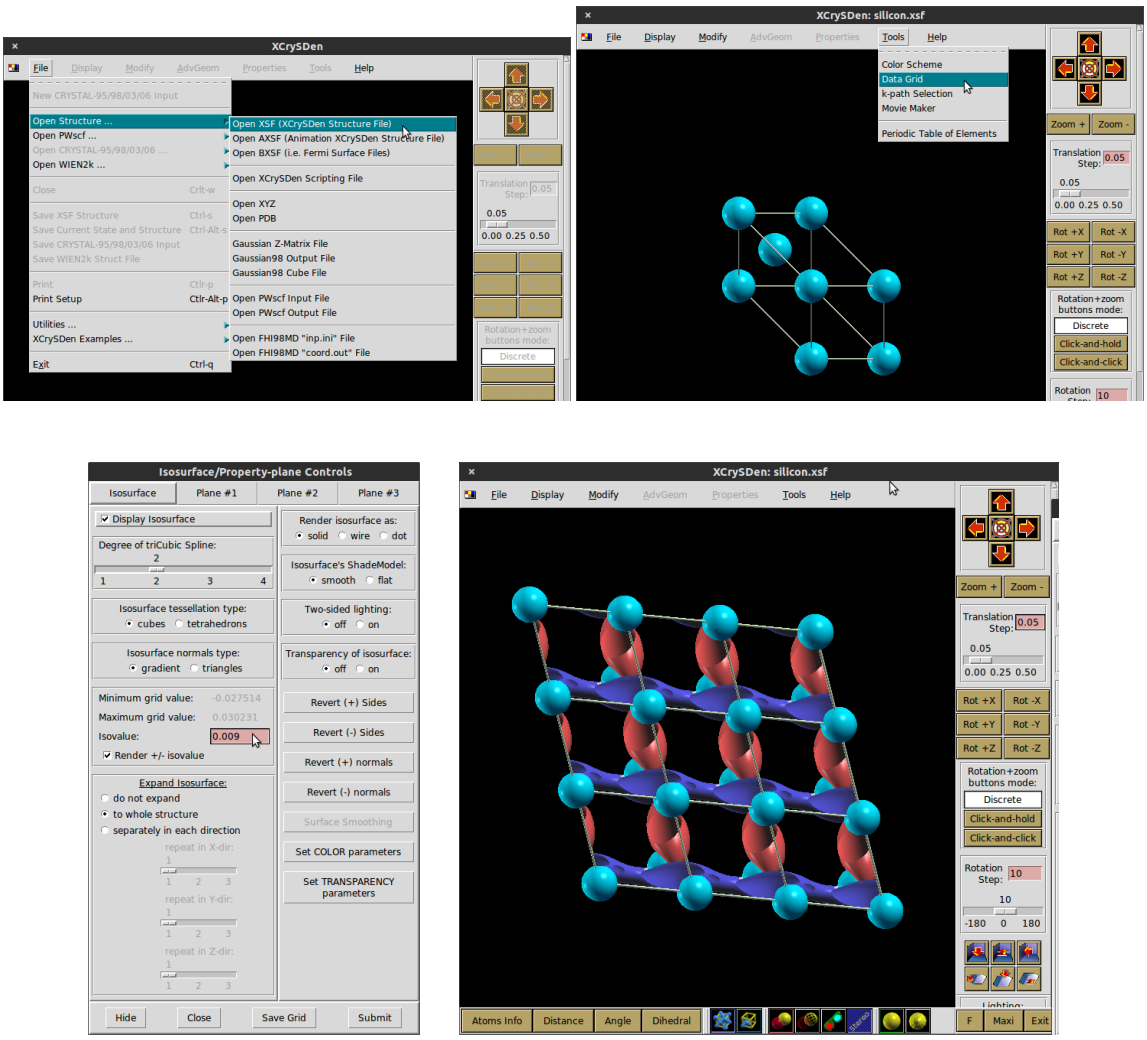
(2) writes the desired quantity to file in a suitable format for various types of plotting and various plotting programs

In order to obtain our wavefunction, first we execute `pw.x` from the previous section, and then we run `pp.x`:

```
$ ibrun -n 4 pw.x < si_nscf.in
$ ibrun -n 4 pp.x < si_pp.in
```

After this operation we should have in our directory the file `silicon.xsf`. We visualize the wavefunctions by launching

`XcrysDen` and following the steps below:



In this example we can see that the electrons at the valence band top concentrate around the Si–Si bonds, as expected from tight-binding models.

▶ Show plots of the 3 highest valence band states of silicon at the Γ point.

▶ Plot the wavefunction of silicon at the conduction band bottom. The conduction band minimum is located along the ΓX line. For the sake of comparison, let us say that the minimum correspond to $\mathbf{k} - (0.85, 0.00, 0.00)2\pi/a$ in in the output file `si_nscf.out`.
Note: In this case you must remove the option `lsign = .true.` from the input file of `pp.x`. Only when $\mathbf{k} = 0$ the wavefunction is real-valued and the sign is well defined, in all other case the wavefunction is a complex quantity.

Self-consistent calculations for metals

Here we will learn how to calculate band structures of metals and their Fermi surfaces. We will use fcc **copper** as an example.

We need a pseudopotential for Cu. As usual, we can select any pseudopotential among those available from http://pseudopotentials.quantum-espresso.org/legacy_tables/original-qe-pp-library. For consistency let us select a pseudopotential based on the DFT/LDA functional:

```
$ wget http://pseudopotentials.quantum-espresso.org/upf_files/Cu.pz-n-van_ak.UPF
```

For the input file we can recycle what we just used for silicon, with only a few modifications:

```
&control
  calculation = "scf",
  prefix = "copper",
  pseudo_dir = "./",
  outdir = "./"
/
&system
  ibrav = 2,
  celldm(1) = 6.678,
  nat = 1,
  ntyp = 1,
  ecutwfc = 40,
  ecutrho = 300,
  occupations = "smearing",
  smearing = "mp",
  degauss = 0.01,
/
&electrons
/
ATOMIC_SPECIES
  Cu 1.0 Cu.pz-n-van_ak.UPF
ATOMIC_POSITIONS alat
  Cu 0.00 0.00 0.00
K_POINTS automatic
8 8 8 1 1 1
```

The parameters `celldm(1)`, `ecutwfc`, `ecutrho`, and `K_POINTS` have been optimized separately. Using these parameters the total energy is converged to better than 5 meV/atom. The new parameter `ecutrho` refers to the charge density, and is needed because we are using a 'Vanderbilt ultrasoft' pseudopotential. In this case we also need to specify a planewaves kinetic energy cutoff for the electron charge density. Typically this cutoff is at least four times larger than our standard `ecutwfc` (because the density involves the square modulus of the wavefunction). In all previous calculations we have been using 'norm-conserving' pseudopotentials, for which `ecutrho` is set by default to four times `ecutwfc`.

The new keywords in red are needed whenever we deal with metals. These keywords instruct `pw.x` that we want to allow for fractional occupations of the Kohn-Sham states (`occupations = "smearing"`), and that occupations are described using a function similar to the Fermi-Dirac distribution (`smearing = "mp"`, 'mp' stands for Methfessel-Paxton), with a width of 0.01 Ry (`degauss = 0.01`).

We test that everything is in place by performing the usual test run:

```
ibrun -n 8 pw.x -npool 8 < copper-1.in > copper-1.out
```

If we now look inside `copper-1.out` and search for

`/` `Fermi`

we find the Fermi level in units of eV:

```
     the Fermi energy is    13.3641 eV
```

This is the energy of the highest occupied Kohn-Sham states.

Band structure of copper

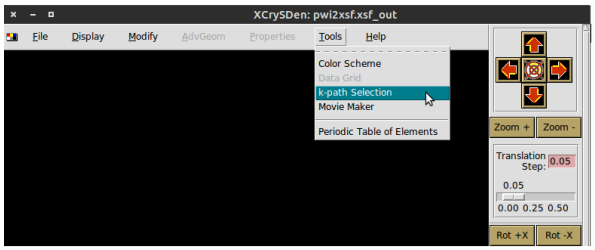
Now we want to calculate the band structure of copper. We proceed in the same way as for silicon:

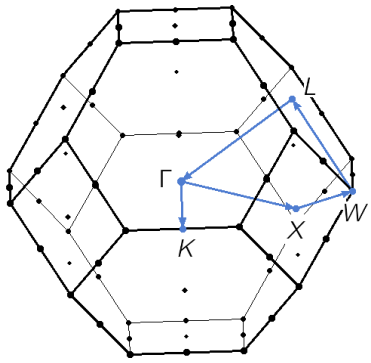
```
$ cat > copper-2.in << EOF
&control
  calculation = "bands"
  prefix = "copper"
  pseudo_dir = "./"
  outdir = "./"
/
&system
  ibrav = 2
  celldm(1) = 6.678
  nat = 1
  ntyp = 1
  ecutwfc = 40
  ecutrho = 300
  occupations = "smearing"
  smearing = "mp"
  degauss = 0.01
  nbnd = 8
/
&electrons
/
ATOMIC_SPECIES
  Cu 1.0 Cu.pz-n-van_ak.UPF
ATOMIC_POSITIONS alat
  Cu 0.00 0.00 0.00
K_POINTS tpiba_b
6
0.00 0.00 0.00 50 ! G
0.00 1.00 0.00 50 ! X
0.50 1.00 0.00 50 ! W
0.50 0.50 0.50 50 ! L
0.00 0.00 0.00 50 ! G
```

```
0.75 0.75 0.00 50 ! K
EOF
```

In this input file we are asking `pw.x` to calculate Kohn-Sham wavefunctions for 8 bands (`nbnd = 8`) along a reciprocal space path with 6 vertices specified in Cartesian coordinates and in units of the lattice parameter (`tpiba_b`). Along each segment we want 50 points. So in this case we will have 50 points from $\Gamma = (0,0,0)$ to $X = (0,1,0)2\pi/a$, and so on.

In general, to decide on the path in the Brillouin zone for the band structure plot we can use `XCrysDen`, which has an option for selecting special points in the Brillouin zone (mind that `XCrysDen` provides special points in reciprocal lattice coordinates):





After executing `pw.x`:

```
$ ibrun -n 8 pw.x -npool 8 < copper-2.in > copper-2.out
```

we use again the small program `bands.x` to extract the eigenvalues along the path. The input file is:

```
$ cat > bands.in << EOF
&bands
  prefix = "copper"
/
EOF
```

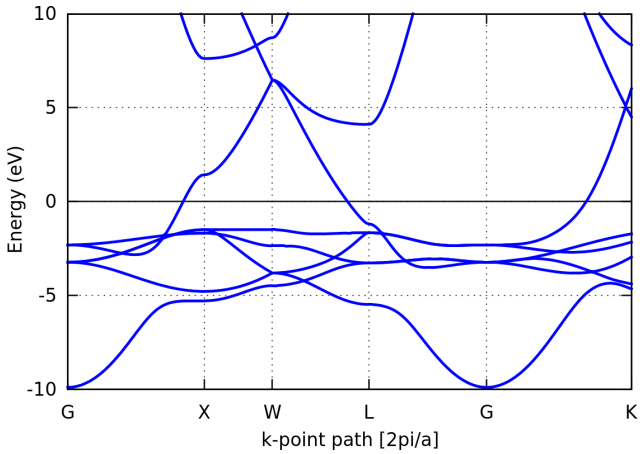
and execute:

```
$ ibrun -n 8 bands.x < bands.in
```

The results will be placed in the file `bands.out.gnu`, and can be plotted using `plot "bands.out.gnu" w l` inside `gnuplot`.

▶ Plot the band structure of Copper, and set the zero of the energy axis to the Fermi energy. The result should look as

follows:



Fermi surface of copper

In this section we want to generate a Fermi surface plot. We will follow the the simplest strategy, which consists of generating Kohn-Sham eigenvalues on a uniform Brillouin zone grid, and using the post-processing utility `fs.x` of Quantum ESPRESSO to convert these data into a format readable by `XCrysDen`. Generally speaking, a Fermi surface plot consists of identifying all the $\mathbf{k}$-points for which $\varepsilon_{n\mathbf{k}}$ is close to the Fermi energy ε_{F} (within a small numerical threshold), and plotting this set of points as a smooth surface.

We create a new input file as follows:

```
$ cat > copper-3.in << EOF
&control
  calculation = "bands"
  prefix = "copper"
  pseudo_dir = "./"
  outdir = "./"
/
&system
  ibrav = 2
  celldm(1) = 6.678
  nat = 1
  ntyp = 1
  ecutwfc = 40
  ecutrho = 300
  occupations = "smearing"
  smearing = "mp"
  degauss = 0.01
  nbnd = 8
/
&electrons
/
```

```
ATOMIC_SPECIES
  Cu 63.546 Cu.pz-n-van_ak.UPF
ATOMIC_POSITIONS alat
  Cu 0.00 0.00 0.00
K_POINTS automatic
30 30 30 0 0 0
EOF
```

The only change with respect to the file `copper-2.in` of the previous section is that we now use an unshifted Brillouin zone grid. This is again a 'non self-consistent calculation', and must follow the standard self-consistent calculation corresponding to the input file `copper-1.in`. Note the much finer grid of points (`30 30 30`), which is needed to produce a smooth surface.

We also need to prepare a simple input file for the post-processing program `fs.x`:

```
$ cp ~/PHY392T/q-e-qe-7.5/bin/fs.x ./
$ cat > fs.in << EOF
&fermi
 prefix = "copper"
/
EOF
```

At this point we can run the calculation:

```
$ ibrun -n 8 pw.x -npool 8 < copper-3.in
$ ibrun -n 8 fs.x < fs.in
```

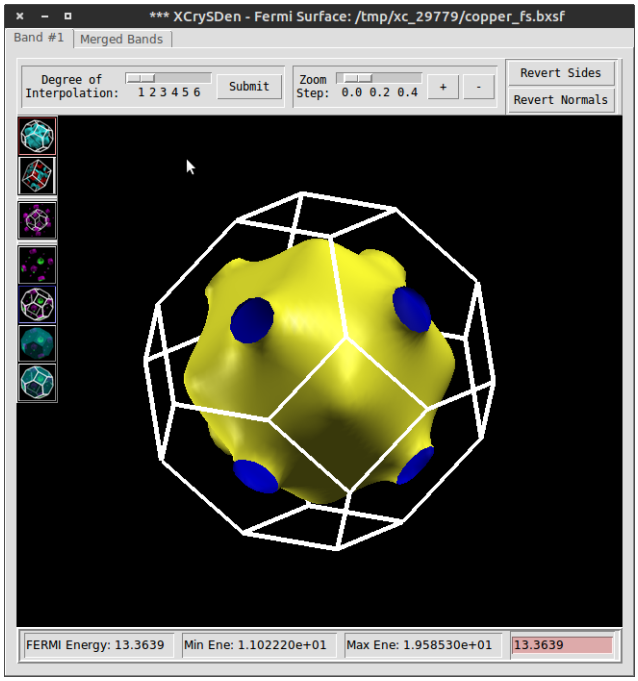
From the output of `fs.x` you can see that this code automatically reads the Fermi energy from the self-consistent run. This information is stored in the file `copper.save/data-file-schema.xml`, along with the electron eigenvalues. `fs.x` reconstructs the eigenvalues on the complete $30 \times 30 \times 30$ Brillouin zone grid starting from the irreducible wedge and using crystal symmetry operations.

The file with the Fermi surface data is `copper_fs.bxsf`. We visualize the Fermi surface by simply executing `XCrysDen` (or alternatively open `XCrysDen` and select `bxsf` format in the input menu):

```
$ xcrysden --bxsf copper_fs.bxsf
```

The visualization of fermi surfaces with `XCrysDen` is very intuitive and does not require much explanation.

▶ Plot the fermi surface of Copper using `XCrysDen`. You should obtain something similar to the following:



Hands-On 10
Dielectric function and optical absorption

In this session we will calculate the dielectric function and the optical absorption spectrum of two semiconductors, **GaAs** and **Si**.

Dielectric function of GaAs

We introduced GaAs in HandsOn 8 for the study of its phonon dispersions and LO-TO splitting. We can use the same input file for the initial, self-consistent field calculation:

```
$ cd $SCRATCH
$ wget https://pseudopotentials.quantum-espresso.org/upf_files/Ga.pz-bhs.UPF
$ wget https://pseudopotentials.quantum-espresso.org/upf_files/As.pz-bhs.UPF
$ cat > GaAs_scf.in << EOF
&control
calculation = "scf"
prefix = "gaas",
pseudo_dir = "./",
outdir = "./"
/
&system
ibrav = 2,
celldm(1) = 10.47494,
nat = 2,
ntyp = 2,
ecutwfc = 40.0,
/
&electrons
/
ATOMIC_SPECIES
Ga 1.0 Ga.pz-bhs.UPF
As 1.0 As.pz-bhs.UPF
ATOMIC_POSITIONS crystal
Ga 0.00 0.00 0.00
As 0.25 0.25 0.25
K_POINTS automatic
6 6 6 1 1 1
EOF
```

We perform a test run to make sure that everything works:

```
$ ibrun -n 8 pw.x -npool 8 < GaAs_scf.in > GaAs_scf.out
```

Now we can take a look at the band structure of GaAs. The procedure is identical to what we did for silicon in HandsOn 9, therefore we can recycle most of the input file `si_nscf.in` from HandsOn 9. The colored lines below indicate the modifications required to work with GaAs instead of Si. These parameters are taken directly from the input file `GaAs_scf.in` above:

```
$ cat > GaAs_bands.in << EOF
&control
 calculation = "bands"
 prefix = "gaas"
 pseudo_dir = "./"
 outdir = "./"
/
&system
 ibrav = 2
 celldm(1) = 10.4749
 nat = 2
 ntyp = 2
 ecutwfc = 40.0
 nbnd = 8
/
&electrons
/
ATOMIC_SPECIES
 Ga 1.0 Ga.pz-bhs.UPF
 As 1.0 As.pz-bhs.UPF
ATOMIC_POSITIONS
 Ga 0.00 0.00 0.00
 As 0.25 0.25 0.25
K_POINTS tpiba_b
 3
0.500 0.500 0.500 20
0.000 0.000 0.000 20
1.000 0.000 0.000 20
EOF
```

We can now run the band structure calculation, precisely as we did for silicon:

```
$ ibrun -np 8 pw.x -npool 8 < GaAs_bands.in > GaAS_bands.out
```

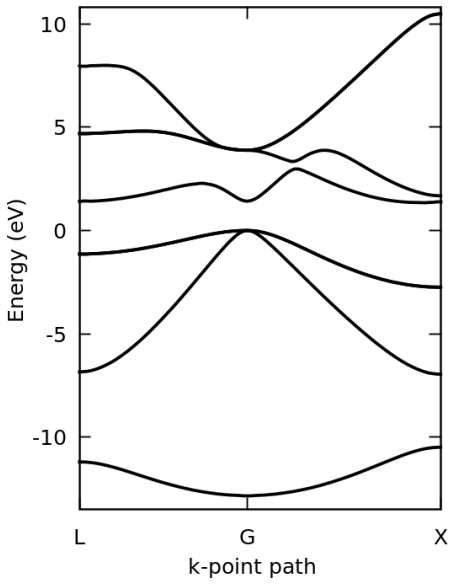
Now we can use the postprocessing tool `bands.x` to extract the Kohn-Sham eigenvalues for each $\mathbf{k}$-point and arrange them in a format to be used in `gnuplot` (remember to copy `bands.x` from your QE bin folder to your current working directory):

```
$ cat > GaAs_bandplot.in << EOF
&bands
 prefix = "gaas"
 outdir = "./"
 filband = "bands"
/
EOF
$ ibrun -n 8 bands.x < GaAs_bandplot.in
```

At this point the file `bands.gnu` contains the band structure data ordered by $\mathbf{k}$-point, in plain text format. This file can be used to plot the bands using `gnuplot` for example:

```
ezero = 5.184
unset key
set ylabel "Energy (eV)"
set xtics ("L" 0, "G" 0.866, "X" 1.866)
set xlabel "k-point path [2pi/a]"
plot "bands.gnu" using 1:($2-ezero) w l lc 0 lw 2 smooth csplines
```

Here we set the top of the valence band at Γ to zero (`ezero = 5.184`). The result looks as follows:



From this band structure we can determine the separation between valence and conduction bands at the three high-symmetry points L, Γ, and X. We find:
$E_L = 1.40$ eV, $E_\Gamma = 1.42$ eV, and $E_X = 1.35$ eV.
The corresponding experimental values are:
$E_L^{\rm exp} = 1.71$ eV, $E_\Gamma^{\rm exp} = 1.42$ eV, and $E_X^{\rm exp} = 1.90$ eV.
We therefore see that, while in the experiment GaAs is a direct-gap semiconductor, with the fundamental (i.e. lowest) band gap at Γ, in the calculations we find a slightly indirect semiconductor with the minimum of the conduction band at X. This is only an artifact of the calculation. In general, the relative energies of the L, Γ, and X gaps are sensitive to the choice of the exchange and correlation functional.

In the following we proceed *as if the calculations gave* GaAs as a direct-gap semiconductor. This assumption is reasonable because in the calculation of the dielectric function below we only consider direct optical transitions, i.e. transitions that conserve $\mathbf{k}$, therefore the smallest gap for direct transitions is precisely E_Γ in our DFT/LDA calculation.

Now we calculate the imaginary part of the dielectric function, $\epsilon_2(\omega)$. This quantity is related to the optical absorption coefficient $\kappa(\omega)$ by $\kappa(\omega) = \omega\,\epsilon_2(\omega)/c\,n(\omega)$, where $\hbar\omega$ is the photon energy, c the speed of light, and n the refractive index.

As we have seen in the theory lectures, the imaginary part of the dielectric function is obtained as:

$$\epsilon_2(\omega) = \frac{\pi\, e^2}{\varepsilon_0 m_e^2 V} \frac{1}{\omega^2} \sum_{cv,\mathbf{k}} |\langle u_{c,\mathbf{k}} | p_x | u_{v,\mathbf{k}} \rangle|^2 \delta(\varepsilon_{c\mathbf{k}} - \varepsilon_{v\mathbf{k}} - \hbar\omega)$$

for light polarized along the x direction. In general we are interested in the isotropic average; however, since GaAs is a cubic compound, each Cartesian direction will give to the same result. The momentum matrix elements in the previous expression are calculated by a post-processing code called `epsilon.x`. We already compiled this program when we issued `make pp`, therefore we only need to copy the code inside the current directory:

```
$ cp ~/PHY392Q/q-e-qe-7.5/bin/epsilon.x ./
```

The manual of this post-processing code can be found in the directory
`~/PHY392Q/q-e-qe-7.5/PP/Doc/eps_man.pdf`.

The input file for `epsilon.x` is as follows:

```
$ cat > GaAs_eps.in << EOF
&inputpp
  outdir = "./"
  prefix = "gaas"
  calculation = "eps"
/
&energy_grid
  smeartype = "gauss"
  intersmear = 0.2
  wmin =  0.0
  wmax = 30.0
  nw = 500
/
EOF
```

This file instructs `epsilon.x` to calculate the real and the imaginary parts of the dielectric function, $\epsilon_1(\omega)$ and $\epsilon_2(\omega)$. The variables `smeartype` and `intersmear` define the numerical approximation used to represent the Dirac delta functions in the expression for ϵ_2 given above. The variables `wmin`, `wmax` and `nw` define the energy grid for the dielectric function. All the energy variables are in eV.

Before executing `epsilon.x` we need to perform a new run with `pw.x`, using a slightly modified input file:

```
$ cat > GaAs_nscf_eps.in << EOF
&control
 calculation = "nscf"
 prefix = "gaas"
 pseudo_dir = "./"
 outdir = "./"
/
&system
 ibrav = 2
```

```
 celldm(1) = 10.4749
 nat = 2
 ntyp = 2
 ecutwfc = 40.0
 nbnd = 16
 nosym = .true.
 noinv = .true.
/
&electrons
/
ATOMIC_SPECIES
 Ga 1.0 Ga.pz-bhs.UPF
 As 1.0 As.pz-bhs.UPF
ATOMIC_POSITIONS
 Ga 0.00 0.00 0.00
 As 0.25 0.25 0.25
K_POINTS automatic
 5 5 5 1 1 1
EOF
```

The modifications brought to our standard input file are as follows:
1. We perform a non-self-consistent calculation
2. We use a uniform grid of **k**-points
3. We turn off the automatic reduction of **k**-points that `pw.x` does by using crystal symmetries (`nosym = .true.` and `noinv = .true.`)
4. We request a larger number of bands (16), since we are interested in interband transitions

The first two modifications are related to the fact that `epsilon.x` is a fairly basic post-processing code and does not recognize crystal symmetries, therefore the entire list of **k**-points in the grid is needed (while by default `pw.x` removes from the list the **k**-points that can be obtained from others by applying symmetry operations).

The grid used in the above input file includes 5^3 (reducible) points.

We can now execute `pw.x` and `epsilon.x`:

```
$ ibrun -n 8 pw.x -npool 8 < GaAs_nscf_eps.in
$ ibrun -n 8 epsilon.x -npool 8 < GaAs_eps.in
```

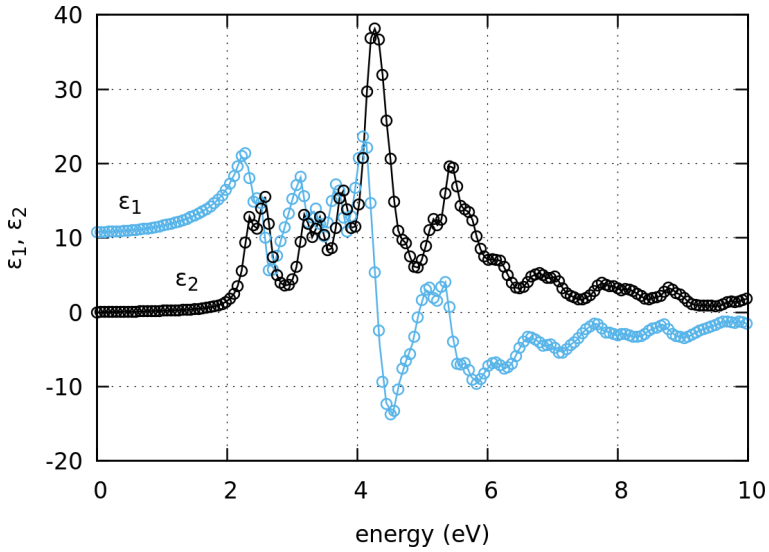
At the end of the execution we will find the output files:

```
$ more epsi_gaas.dat

# energy grid [eV]     epsi_x  epsi_y  epsi_z
#
    0.000000000    0.000000000    0.000000000    0.000000000
    0.060120240    0.008731751    0.008731751    0.008731751
...

$ more epsr_gaas.dat
```

```
# energy grid [eV]     epsr_x  epsr_y  epsr_z
#
    0.000000000   10.460661055   10.460660612   10.460662360
    0.060120240   10.806301744   10.806301683   10.806301696
...
```

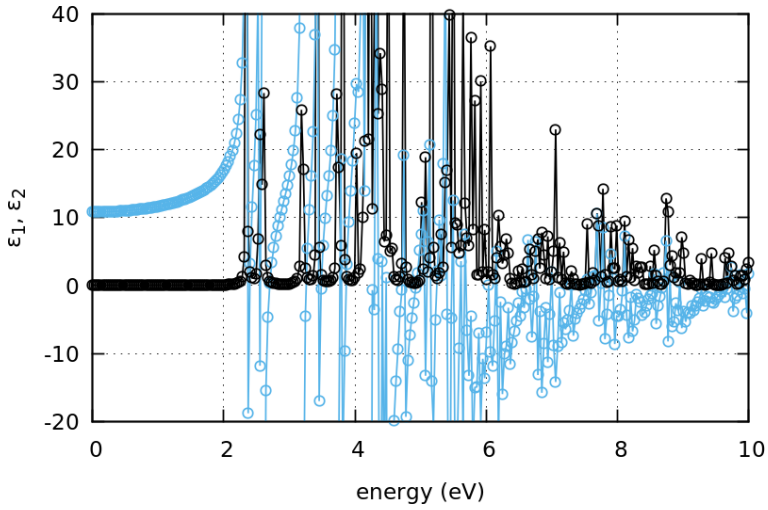
The first file contains the real part of the dielectric function, for an electric field polarized along x, y, or z. The second file is the corresponding imaginary part. the x, y and z components are identical.

A plot of these quantities using `gnuplot` yields:



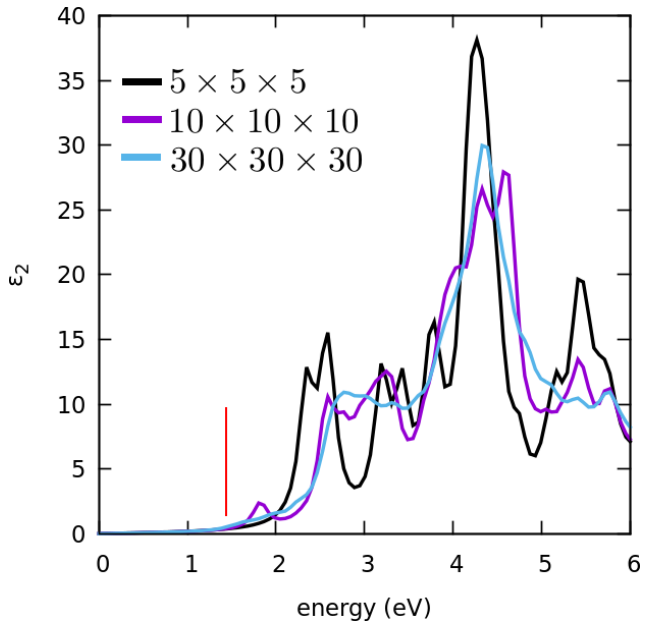
We can see that the curves are not very smooth. This phenomenon is related to the sampling of the Brillouin zone: in this calculation we used a $5 \times 5 \times 5$ mesh, therefore we are trying to perform a continuous integral over the Brillouin zone using a discrete grids that contains only 125 points.

To better see the effect of the discretization of the Brillouin zone, we can reduce the width of the Gaussian used to replace the Dirac delta functions in ϵ_2. As an example the plot below was obtained by setting `intersmear = 0.01` (eV) in `gaas_eps.in`:



Here we can clearly see that we have discrete transitions because we only have 4 valence bands, 12 conduction bands, and 125 **k**-points. By using a larger `intersmear` parameter we can make this curve smoother, but by doing so we loose the fine features of the spectrum.

In actual calculations, the meshes required to obtain converged dielectric functions may need to contain as many as $100 \times 100 \times 100$ points. Below we compare ϵ_2 for the cases of $5 \times 5 \times 5$, $10 \times 10 \times 10$ points, and $30 \times 30 \times 30$ points, using `intersmear = 0.2` (eV). We see that the curve is starting to converge around $30 \times 30 \times 30$, but some improvement is still needed:

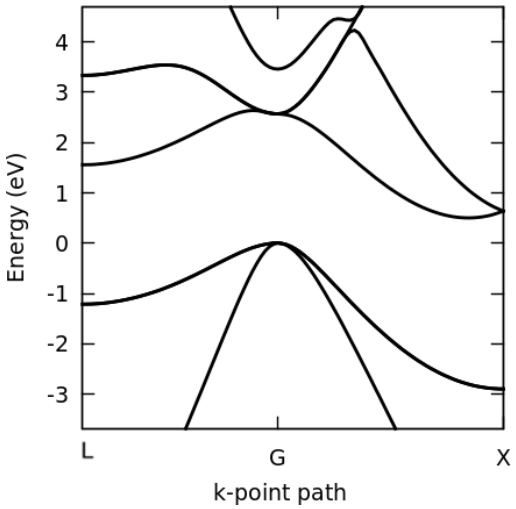


In this figure the red vertical line indicates the direct gap, 1.41 eV. We see that ϵ_2 vanishes below this threshold (except for the residual tails of the Gaussian smearing). This is the **onset of optical absorption**, because the absorption coefficient is proportional to ϵ_2.

Dielectric function of Si

Now we repeat the calculations performed in the previous section for the case of silicon.

We already studied the band structure of silicon in HandsOn 9, so we reproduce a close-up of the band structure below:



Since the smallest direct gap is at Γ, $E_\Gamma = 2.57$ eV (the experimental value is 3.5 eV), we expect ϵ_2 to vanish below this energy. Let us check this hypothesis.

We use the same setup as in HandsOn 9 for the self-consistent calculation:

```
$ wget https://pseudopotentials.quantum-espresso.org/upf_files/Si.pz-vbc.UPF
$ cat > si_scf.in << EOF
&control
 calculation = "scf"
 prefix = "silicon"
 pseudo_dir = "./"
 outdir = "./"
/
&system
 ibrav = 2
 celldm(1) = 10.2078
 nat = 2
 ntyp = 1
 ecutwfc = 25.0
/
&electrons
/
ATOMIC_SPECIES
 Si 28.086 Si.pz-vbc.UPF
ATOMIC_POSITIONS
 Si 0.00 0.00 0.00
 Si 0.25 0.25 0.25
```

```
K_POINTS automatic
 4 4 4 1 1 1
EOF
$ ibrun -n 8 pw.x -npool 8 < si_scf.in
```

For the non-self-consistent calculation on a uniform Brillouin zone grid, we recycle the input file for GaAs used in the previous section, and we adapt it to the case of silicon:

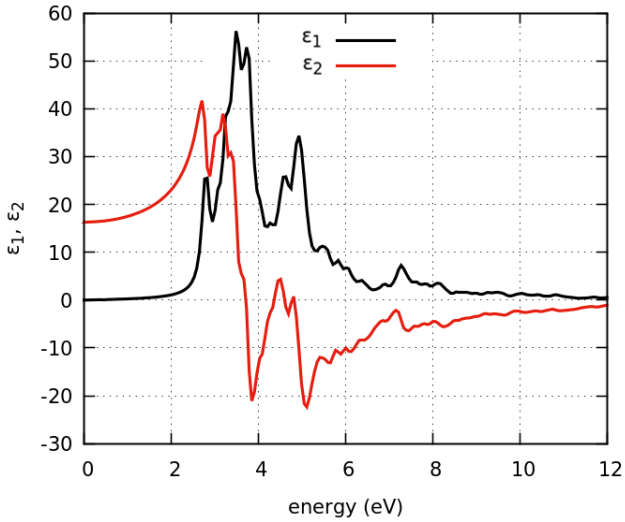
```
$ cat > Si_nscf_eps.in << EOF
&control
 calculation = "nscf"
 prefix = "silicon"
 pseudo_dir = "./"
 outdir = "./"
/
&system
 ibrav = 2
 celldm(1) = 10.2078
 nat = 2
 ntyp = 1
 ecutwfc = 25.0
 nbnd = 16
 nosym = .true.
 noinv = .true.
/
&electrons
/
ATOMIC_SPECIES
 Si 28.086 Si.pz-vbc.UPF
ATOMIC_POSITIONS
 Si 0.00 0.00 0.00
 Si 0.25 0.25 0.25
K_POINTS automatic
 5 5 5 1 1 1
EOF
```

Similarly, we adapt the input file for `epsilon.x` already used for GaAs:

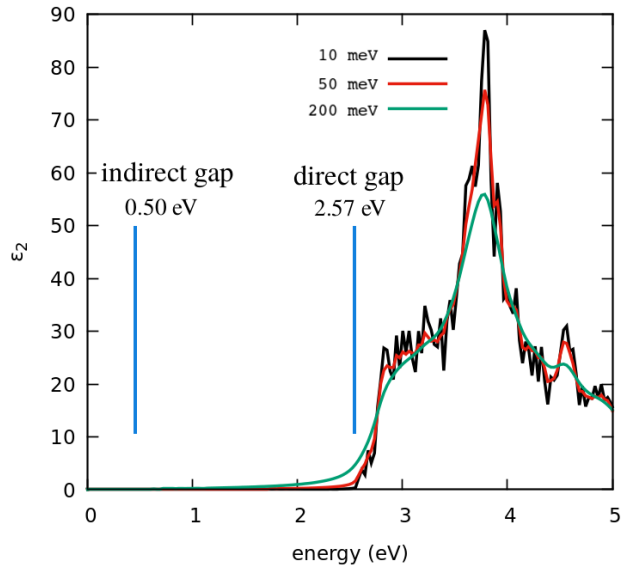
```
$ cat > Si_eps.in << EOF
&inputpp
  outdir = "./"
  prefix = "silicon"
  calculation = "eps"
/
&energy_grid
 smeartype = "gauss"
```

```
 intersmear = 0.2
 wmin =  0.0
 wmax = 30.0
 nw = 500
/
EOF
$ ibrun -n 8 epsilon.x -npool 8 < Si_eps.in
```

The following plot shows the dielectric function obtained using these settings:



A more refined calculation using $30\times30\times30$ points is shown below, using a Gaussian smearing of 10 meV (`intersmear = 0.01`), 50 meV (`intersmear = 0.05`), and 200 meV (`intersmear = 0.2`), respectively.



In this plot we see that, as expected, the onset of optical absorption coincides with the direct gap (2.57 eV). While the

minimum gap is 0.50 eV in our calculations (1.1 eV in experiments), ϵ_2 vanishes between 0.50 eV and 2.57 eV because optical transitions in this energy range do not fulfill momentum conservation.

In order to correctly describe the experimentally-observed optical transitions in the energy range between the indirect gap and the direct gap, we must include **phonon-assisted transitions** in the theory. This extra step is feasible but it would require us to cover additional theory which is outside of the scope of our introuctory course.