

Connecting to the Maryland Advanced Research Computing Center (MARCC)

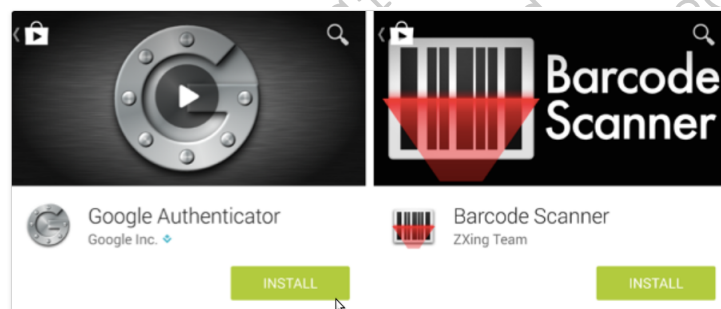
Two-factor authentication

In order to connect to MARCC we need to setup the 'two-factor authentication' protocol.

For this you will need:

- The [username](#) sent to you by MARCC via email
- The temporary [password](#) sent to you by MARCC via email
- The Google Authenticator

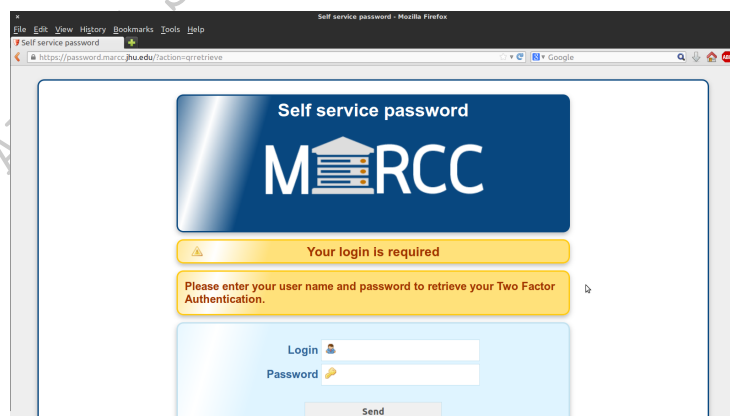
The Google Authenticator can be installed from the Play Store on your mobile device. You will also need a Barcode Scanner. These two apps look like this:



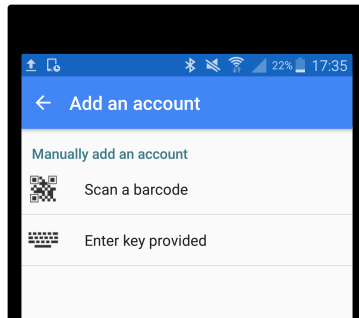
Now go to this page from your laptop:

<https://password.marcc.jhu.edu/?action=qrretrieve>

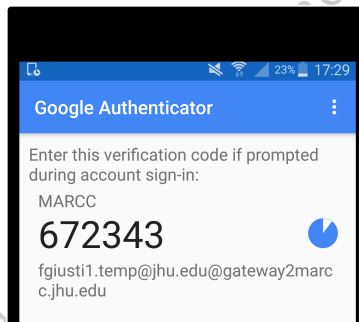
You will be prompted for username and password, here we need to enter what we received by email:



At this point we link the Google Authenticator with MARCC using the barcode provided:



The above steps are performed only once. From now on we can login into MARCC by using our [username](#), [password](#), and [verification code](#). The verification code is the number provided by the Google Authenticator, as in the following snapshot:



The standard procedure for logging in is discussed in Tutorial 1.1.

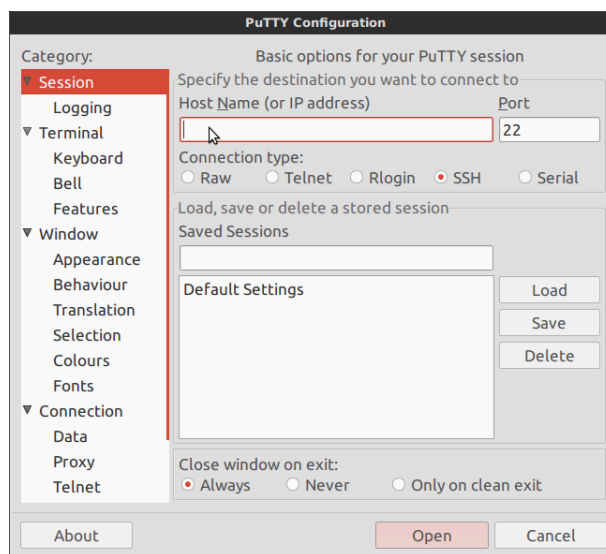
Connecting from a Windows laptop

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

A popular choice is [Putty](http://www.putty.org). This can be downloaded from:

<http://www.putty.org>

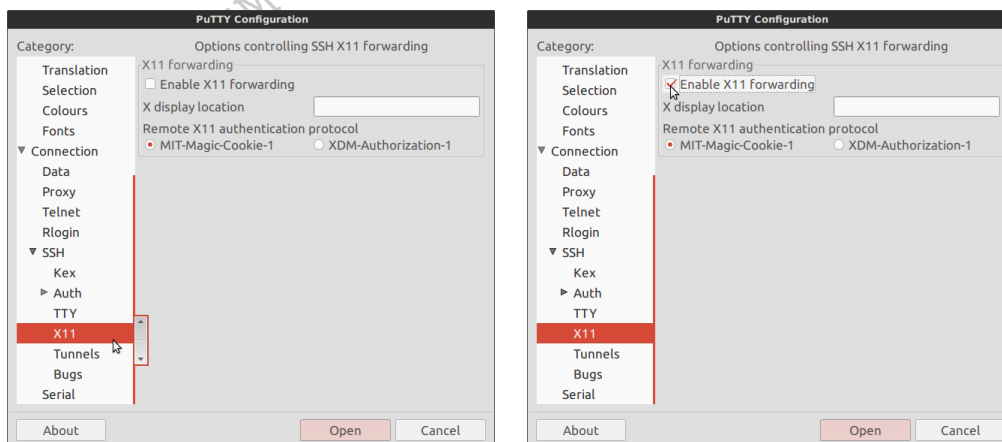
When you execute Putty you will see something like the following:



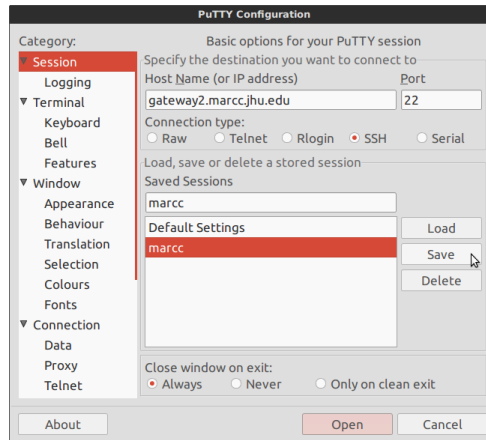
In the field 'Host Name' we enter:

`gateway2.marcc.jhu.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 'marcc':



Visualizing graphics from a Windows laptop

In order to visualize graphics when using Putty, your laptop 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 Xming is installed, the procedure for running calculations and visualizing graphics on MARCC is as follows:

- We launch Xming. This application will now run in the background.
- We execute Putty.

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

An introduction to density functional theory for experimentalists

Tutorial 1.1

Login shell and compilation

In order to work on the HPC cluster we need to establish a secure connection. We first open a terminal, and then type:

```
$ ssh -X gateway2.marcc.jhu.edu -l fgiusti1.temp@jhu.edu
```

where `fgiusti1.temp@jhu.edu` must be replaced by the username that you have been assigned. After entering your password you will see something like:

```
[fgiusti1.temp@jhu.edu@login-node04 ~]$
```

This is the command line that we will use from now on.

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 picky with spaces):

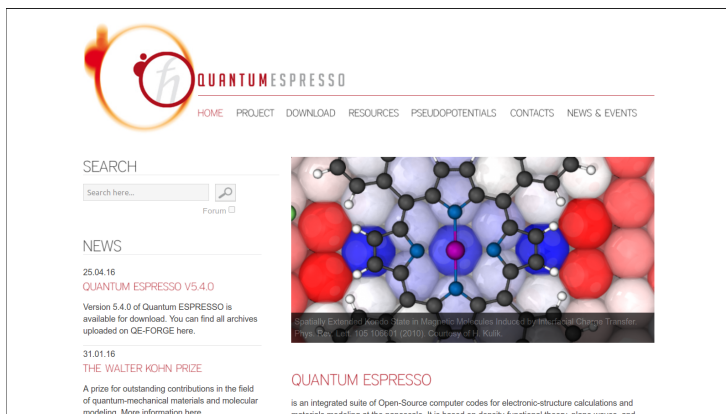
```
cat >> .bashrc << EOF
PS1="$ "
alias c="clear"
alias l="ls -lh"
module load fftw3/intel/3.3.4
module load xcrysden
module load gnuplot/5.0.0
module load openmpi/intel/1.8.4
EOF
source ~/.bashrc
```

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

We can now create our working directory for this school:

```
$ mkdir scratch/summerschool ; cd ~/scratch/summerschool
```

In this school we will be using the Quantum ESPRESSO (QE) software package. The website can be found at www.quantum-espresso.org



In order to use QE we download the package and we install it in our working directory. We first download the latest release of QE as a zipped archive:

```
$ wget http://www.qe-forge.org/gf/download/frsrelease/211/968/espresso-5.4.0.tar.gz
```

Then we unpack and remove the zipped archive:

```
$ tar xzf espresso-5.4.0.tar.gz ; rm espresso-5.4.0.tar.gz
```

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

```
$ cd espresso-5.4.0 ; l
```

```
total 128K
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 archive
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 clib
-rwxr-xr-x 1 fgiusti1.temp@jhu.edu paradim 2.3K Apr 24 17:18 configure
drwxr-xr-x 6 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 COUPLE
drwxr-xr-x 5 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 CPV
drwxr-xr-x 3 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 dev-tools
drwxr-xr-x 6 fgiusti1.temp@jhu.edu paradim 12K Apr 24 17:23 Doc
-rw-r--r-- 1 fgiusti1.temp@jhu.edu paradim 3.4K Apr 24 17:19 environment_variables
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 FFTXlib
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 include
drwxr-xr-x 3 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 install
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 LAXlib
-rw-r--r-- 1 fgiusti1.temp@jhu.edu paradim 18K Apr 24 17:19 License
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 LR_Modules
-rw-r--r-- 1 fgiusti1.temp@jhu.edu paradim 13K Apr 25 13:19 Makefile
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 12K Apr 24 17:20 Modules
drwxr-xr-x 7 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 PP
drwxr-xr-x 2 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 pseudo
drwxr-xr-x 6 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 PW
-rw-r--r-- 1 fgiusti1.temp@jhu.edu paradim 1.3K Apr 24 17:19 README
drwxr-xr-x 3 fgiusti1.temp@jhu.edu paradim 4.0K Apr 24 17:19 upftools
```

At the beginning of this school we will be mainly interested in the program `pw.x`, which is contained in the directory `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 the compilers and numerical libraries. This information is determined by the program `configure`. Therefore we issue:

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

This operation will require approximately 10 min.

While we wait we may as well check the page of MARCC where the cluster [Bluecrab](#) is described: [Maryland Advanced Research Computing Center](#).

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

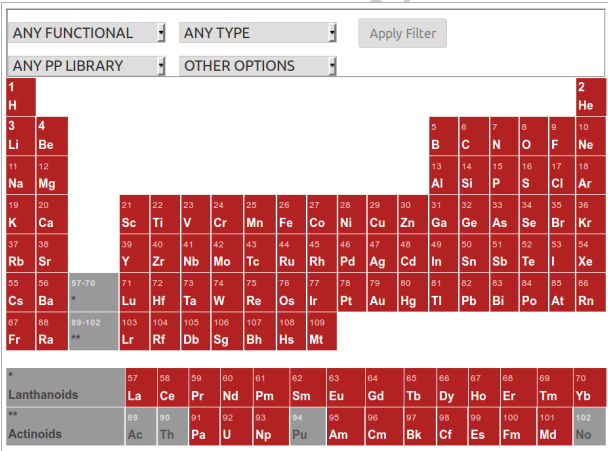
```
$ ls -l bin
```

```
lrwxrwxrwx 1 fgiustil.temp@jhu.edu paradim 14 Jun 28 18:35 pw.x -> ../PW/src/pw.x
```

Test run

Now we want to prepare a very simple job and execute it 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 silicon in the diamond structure. In order to proceed we first need a 'pseudopotential'. Pseudopotentials will be introduced formally in Lecture 2.2, for now it suffices to know that for each atom we will need one pseudopotential. The QE pseudopotential libraries can be found at <http://www.quantum-espresso.org/pseudopotentials>. By clicking on this link we find the following periodic table



The image shows a screenshot of the Quantum Espresso pseudopotential website. At the top, there are several dropdown menus: 'ANY FUNCTIONAL', 'ANY TYPE', 'ANY PP LIBRARY', and 'OTHER OPTIONS', along with an 'Apply Filter' button. Below the filters is a periodic table of elements. The elements are color-coded: red for main group elements, blue for transition metals, and grey for lanthanoids and actinoids. The table includes element symbols and atomic numbers. A watermark 'Giustino, Cornell, July 2016' is visible across the table.

Now we can click on silicon and we will see a list of available pseudopotentials. In principle we could download the entire library once and for all, but for now let us proceed on a case-by-case basis. 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 on the HPC cluster:

```
$ cd ..
```

```
$ mkdir tutorial-1.1 ; cd tutorial-1.1
```

```
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/Si.pz-vbc.UPF
```

Now we should have the pseudopotential file inside the directory `tutorial-1.1`. We can see inside this plain text file by using the command `more`:

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

```

<PP_INFO>
Generated using unknown code
Author: von Barth and Car   Generation date: before 1984
Info: Si LDA 3s2 3p2 VonBarth-Car, l=2 local
    0           The Pseudo was generated with a Non-Relativistic Calculation
0.000000000000E+00   Local Potential cutoff radius
nl pn  l  occ           Rcut           Rcut US           E pseu
3S  0  0  2.00       0.000000000000       0.000000000000       0.000000000000
3P  0  1  2.00       0.000000000000       0.000000000000       0.000000000000
</PP_INFO>

```

```

<PP_HEADER>
    0           Version Number
    Si          Element
    NC          Norm - Conserving pseudopotential
    F           Nonlinear Core Correction
SLA PZ  NOGX NOGC   PZ  Exchange-Correlation functional
4.000000000000     Z valence
0.000000000000     Total energy
0.00000000  0.00000000 Suggested cutoff for wfc and rho
    1           Max angular momentum component
431              Number of points in mesh
    2    2       Number of Wavefunctions, Number of Projectors
Wavefunctions     nl  l  occ
3S  0  2.00
3P  1  2.00
</PP_HEADER>

```

```

<PP_MESH>
<PP_R>
1.30825992062E-03  1.34137867819E-03  1.37533584110E-03  1.41015263368E-03
1.44585081756E-03  1.48245270526E-03  1.51998117417E-03  1.55845968079E-03
...

```

For simplicity we also copy the executable inside the current working directory (this is not standard practice but it makes things easier to understand the first time):

```
$ cp ../espresso-5.4.0/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, and the job submission script for the HPC cluster. 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,
cellldm(1) = 10.28,
nat = 2,
ntyp = 1,
ecutwfc = 18.0,
/

```

```
&electrons
conv_thr = 1.0d-8
/
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
```

For the job submission script we can create `job-1.pbs`:

```
$ cat << EOF > job-1.pbs
#!/bin/bash -l
#SBATCH --reservation=Paradim
#SBATCH --job-name=job-1
#SBATCH --time=00:30:00
#SBATCH --partition=parallel
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=12
#SBATCH --mem-per-cpu=2000MB
mpirun -np 12 pw.x -npool 4 < silicon-1.in > silicon-1.out
EOF
```

In the current directory we should now see the following:

```
$ ls
job-1.pbs  pw.x  silicon-1.in  Si.pz-vbc.UPF
```

Eventually we are ready to submit our first job to the queue. We issue:

```
$ qsub job-1.pbs
```

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

```
$ whoami
fgiusti1.temp@jhu.edu
```

```
$ qstat -u fgiusti1.temp@jhu.edu
```

```
login-node04.cm.cluster:
Job id          Username Queue   Name      SessID NDS  TSK  Req'd Req'd  Elap
-----
7415791        fgiusti1 shared  job-1    --      1   12   --    00:30 Q 00:00
```

If the cluster is too busy we can alternatively try to run within an **interactive** session:

```
$ interact-paradim -r Paradim -p parallel -n 12 -c 1 -t 60 -m 24G
```

This command opens a session where we will be able to execute `pw.x` directly from the command line, ie without a job submission script.

Here we are requesting 12 CPUs, since `-n 12` is the number of tasks, and `-c 1` is the number of cores per task. We are asking for a session of 60 min (`-t 60`) and with a total memory of 24 GB (`-m 24G`).

If we enter an interactive session, then we can simply run `pw.x` by issuing:

```
$ mpirun -n 12 pw.x -npool 4 < silicon-1.in > silicon-1.out
```

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

```
$ more silicon-1.out
Program PWSCF v.5.4.0 starts on  8Jul2016 at  8:27:35

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);
  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  12 processors
K-points division:      npool      =      4
R & G space division:  proc/nbgrp/npool/nimage =      3
Waiting for input...
Reading input from standard input
...
PWSCF      :      0.11s CPU      0.18s WALL

This run was terminated on:  8:27:35  8Jul2016

-----
JOB DONE.
-----
```

Generating new jobs

Throughout this school we will prepare input files and job submission scripts by modifying the files `silicon-1.in` and `job-1.pbs`.










Instead of using the `cat` command as we did in the previous section, we first create two new files by just copying the previous ones:

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

Now we use `vi` to modify the newly-created files:

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

This 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  and modify as we wish
 - 3 When we are done making changes we press 
 - 4 We write the modified file and exit by typing   

As an example we now change the parameter `ecutwfc` from 18 Ry to 30 Ry. This parameter represents the planewaves kinetic energy cutoff and will be described in Lecture 2.2. 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.

In the case of `job-2.pbs` we proceed similarly. We first change the input and output filenames into `silicon-2.in` and `silicon-2.out`. We also change the number of CPUs to be used from 4 (`-np 4`) to 12 (`-np 12`).

We submit this job as we did earlier:

```
$ qsub job-2.pbs
```

During the hands-on session we will explore in detail various calculation parameters and runtime options.

Documentation

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

http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PW.html

During the next few days we will also need documentation for the code `ph.x` (phonons) and `pp.x` (post-processing). These can be found here:

`pp.x`: http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PH.html

`pp.x`: http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PP.html

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

<http://www.marcc.jhu.edu/getting-started/running-jobs>

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

```
$ sinfo -s
```

An introduction to density functional theory for experimentalists

Tutorial 1.2

Hands-on session

Exercise 1

► Repeat the steps illustrated during Tutorial 1.1, in particular:

- 1 Login into your account, set the modules in the file `.bashrc`, and create your working directory
- 2 Download the QE software 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` and submission script `job-1.pbs`
- 5 Submit this job and check the output file `silicon-1.out`

For all these steps you can directly copy/paste the instructions from the PDF document of Tutorial 1.1 (or type everything if you are patient).

Exercise 2

We want to explore one important convergence parameter of DFT calculations, the planewave kinetic energy cutoff `ecutwfc`.

To this aim we create a new directory:

```
$ cd ~/scratch/summerschool ; mkdir tutorial-1.2 ; cd tutorial-1.2
```

and we copy over the important files generated in the previous exercise:

```
$ cp ../tutorial-1.1/pw.x ./
$ cp ../tutorial-1.1/Si.pz-vbc.UPF ./
$ cp ../tutorial-1.1/silicon-1.in ./silicon-3.in
$ cp ../tutorial-1.1/job-1.pbs ./job-3.pbs
```

Using `vi` we modify the input variable `ecutwfc` to 5 Ry (note 1 Ry = 13.6058 eV). After this change, line #23 of `silicon-3.in` should read:

```
ecutwfc = 5.0,
```

In order to be consistent with the new names of the input file we must also modify the job submission script `job-3.pbs` using `vi`, so as to have `silicon-3.in` and `silicon-3.out`.

Now we submit the job to the cluster as usual:

```
qsub job-3.pbs
```

When the job is finished we can analyze the output file `silicon-3.out`. This output file contains the most important information regarding your run. Throughout this school we will learn the meaning of the various sections of this file gradually. For now we concentrate only on a few simple aspects.

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 in Lecture 5.2). The numbers are internal codes of `pw.x`.

Now we search for the words `kinetic-energy cutoff`. This should be identical to the value of `ecutwfc` set in the input file. This parameter is the kinetic energy cutoff of the planewaves basis set, and will be introduced formally in Lecture 2.2. This parameter sets the number of planewaves in which every Kohn-Sham wavefunction is expanded (ie the number of Fourier components of each wavefunction). The number of planewaves corresponding to the cutoff `ecutwfc` can be found by searching for `Kohn-Sham Wavefunctions`. You will see the following line:

```
Kohn-Sham Wavefunctions    0.00 Mb ( 58, 4)
```

This means that we have 4 Kohn-Sham wavefunctions (corresponding to the 4 valence bands of silicon), and that each wavefunction is expressed as a linear combination of 58 planewaves.

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 = -15.60437814 Ry
```

This value should be taken with caution: it contains an offset which arises from the use of pseudopotentials (see Lecture 2.2), and it is not referred to vacuum, since there is no vacuum reference when we perform a calculation in a infinitely-extended crystal. This means that the absolute value of DFT total energies in extended solids is not meaningful; what is meaningful is the **total energy difference** between two configurations.

Finally we look at the timing: search for the word `'PWSCF :'` (mind the colon and the blanks). You should find:

```
PWSCF : 0.08s CPU 0.20s 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.

Now we want to study how the total energy, the number of planewaves, and the timing 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 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. You can collect the results for example by creating a text file using `vi exercise2.txt` and entering your results one by one. You should be able to construct a file looking like this [please note that your numbers will not be identical since this was executed on a different cluster]:

```
$ more excercise2.txt
```

```
# ecutwfc (Ry)   planewaves   energy (Ry)   time (s)
   5             58       -15.60437814  0.07
  10            153       -15.77558550  0.10
  15            274       -15.83422234  0.10
  20            416       -15.84721988  0.16
  25            580       -15.85087570  0.18
  30            763       -15.85182962  0.19
  35            959       -15.85235153  0.23
  40           1185       -15.85268618  0.27
```

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

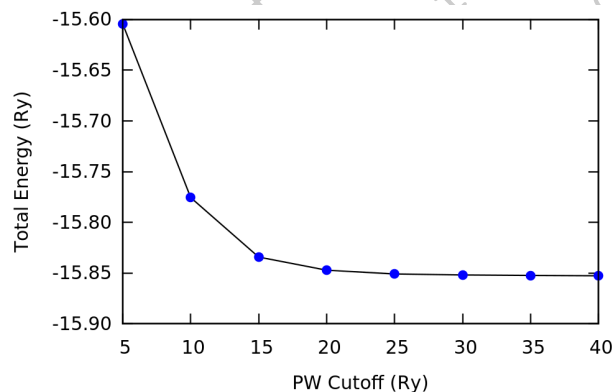
```
$ cat > loop.tcsh << EOF
foreach ECUTWFC ( 5 10 15 20 25 30 35 40 )
  sed "s/5\\.0/\\${ECUTWFC}/g" silicon-3.in > silicon-\\${ECUTWFC}.in
end
EOF
$ tcsh loop.tcsh
```

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                =   -15.60437814 Ry
```

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 `exercise2.txt` using the command `scp` or the program `filezilla`, and then plot the data using your favourite software (eg Origin or Excel).

For the total energy you should find something like this:



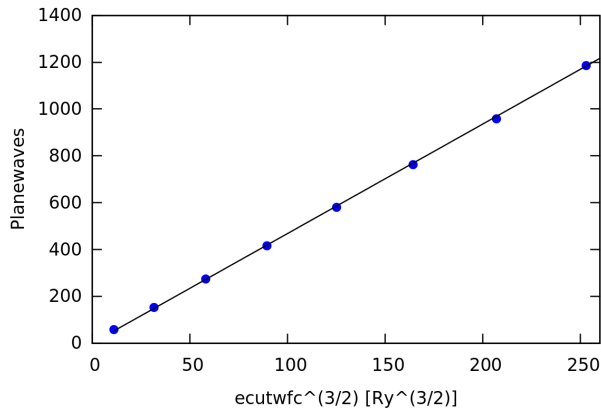
Here we see that, using a cutoff of 25 Ry, we obtain a total energy which is only 12 meV/atom higher than our best-converged value at 40 Ry. In principle we should test even higher values of `ecutwfc` (the correct result is obtained in the limit of this cutoff reaching infinity), but in practice 25 Ry is good enough for this tutorial. Most quantities that can be computed using DFT depend critically on this cutoff, therefore it is **very important** to always perform this test when running DFT calculations.

If the planewaves cutoff is so important, why 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. You can test this directly by plotting the CPU time vs. the cutoff using the values inside the file `exercise2.txt`.

In this example the runtime is below 1 sec, therefore the choice of the cutoff is not important in practice. However, in most DFT calculations a careful choice of cutoff can save us weeks of computer time.

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

- ▶ Verify that the number of planewaves increases with the cutoff.
- ▶ Verify that a plot of the number of planewaves vs. $\text{ecutwfc}^{3/2}$ yields a straight line:



- ▶ Can you explain the origin of this relation between the cutoff and the number of planewaves?
Note: in order to answer this question you will need to go through Lecture 2.2 first.

Exercise 3

We now want to explore one other convergence parameter of DFT calculations for crystals, the Brillouin zone sampling `K_POINTS` (to be described in Lecture 2.2).

In the input file `silicon-3.in` we had requested a uniform sampling of Bloch wavevectors \mathbf{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 $4 \times 4 \times 4 = 64$ \mathbf{k} -vectors.

- ▶ Now search for 'number of k points' in the output file `silicon-4.in`. You should find:

```
number of k points= 10
```

Therefore the code is using only 10 \mathbf{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 \mathbf{k} -points, using the same procedure as in Exercise 2. 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`. For these calculations you can use our 'converged' cutoff `ecutwfc = 25.0 Ry`.

Note In this case you will need to set the execution flag `-npool` inside the submission script to 1, for example: `-n 8 pw.x -npool 1` instead of the original `-n 4 pw.x -npool 4`.

- ▶ 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 following ones:

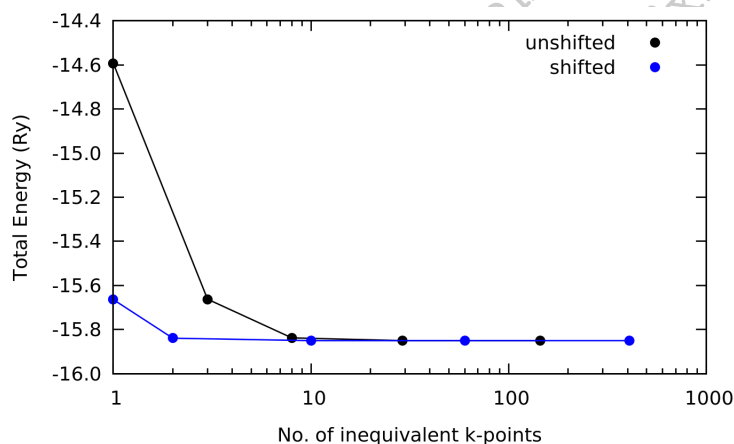
```
$ more excercise3a.txt
```

```
# grid      shift      energy (Ry)  inequiv. k-points  time (s)
  1  1  1    0 0 0   -14.59239650      1             0.20
  2  2  2    0 0 0   -15.66198445      3             0.25
  4  4  4    0 0 0   -15.83707680      8             0.21
  8  8  8    0 0 0   -15.85079066     29            0.56
 16 16 16    0 0 0   -15.85108168    145            2.41
```

```
$ more excercise3b.txt
```

```
# grid      shift      energy (Ry)  inequiv. k-points  time (s)
  1  1  1    1 1 1   -15.66368666      1             0.10
  2  2  2    1 1 1   -15.83894845      2             0.10
  4  4  4    1 1 1   -15.85087570     10            0.22
  8  8  8    1 1 1   -15.85108292     60            1.04
 16 16 16    1 1 1   -15.85108131    408            6.88
```

► 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, < 2 meV/atom away from our best value at 8 8 8 1 1 1. We also see that the shifted grid converges faster 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.

Exercise 4

In this exercise we want to explore the **scaling** of DFT calculations 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:

```
$ wget http://giustino.materials.ox.ac.uk/tutorial_1.2_exercise_4.tgz
$ tar xzf tutorial_1.2_exercise_4.tgz ; ls tutorial_1.2_exercise_4
```

```
silicon-4.1.in  silicon-4.2.in  silicon-4.3.in  silicon-4.4.in  silicon-4.5.in
```

These files correspond to supercells of silicon, containing one primitive unit cell (`silicon-4.1.in`), a $2 \times 2 \times 2$ supercell (`silicon-4.2.in`), and so on, up to $5 \times 5 \times 5$ primitive unit cells (`silicon-4.5.in`). By looking inside these input files you can check that we have a number of Si atoms ranging from 2 to 250.

► Now run `pw.x` using these five different 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 will depend on the cluster, but the relative times are meaningful)

# atoms	cputime (s)
2	0.09
16	0.44
54	4.53
128	34.82
250	218.78

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

Generally speaking DFT calculations scale with the cube of the number of atoms, $T_{\text{CPU}} = \text{const} \cdot N^3$ (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 here is that if we double the size of our system, then our DFT calculation will require approximately 8 times longer to complete (eg 1 week \rightarrow 8 weeks).

An introduction to density functional theory for experimentalists

Tutorial 2.1

As usual we create a new folder on the HPC cluster:

```
$ cd ~/scratch/summerschool ; mkdir tutorial-2.1 ; cd tutorial-2.1
```

Equilibrium structure of a diatomic molecule

In this tutorial we are going to learn how to calculate the equilibrium structures of simple systems. The formal theory required for these calculations will be discussed in Lecture 3.1, for now we can just use the following rule of thumb:

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

The total potential energy is the same quantity that we have been using during Tutorial 1.1 and Tutorial 1.2 (eg when we did `grep "\!" silicon-1.out`). This quantity includes all terms of the electron-ion Hamiltonian, except the kinetic energy of the ions.

Let us calculate the equilibrium structure of the Cl_2 molecule.

The Cl_2 molecule has only 2 atoms, and the structure is completely determined by the Cl–Cl bond length. Therefore we can determine the equilibrium structure by calculating the total energy as a function of the Cl–Cl distance.

The first step is to find a suitable pseudopotential for Cl. As in Tutorial 1.1 we go to <http://www.quantum-espresso.org/pseudopotentials> and look for Cl. We recognize LDA pseudopotential by the label pz in the filename. Let us go for the following:

```
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/Cl.pz-bhs.UPF
```

We also copy the executable, job submission script, and input file from the previous tutorial:

```
$ cp ../tutorial-1.1/pw.x ./
$ cp ../tutorial-1.1/job-1.pbs ./
$ cp ../tutorial-1.1/silicon-1.in ./cl2.in
```

Now we can 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
conv_thr = 1.0d-8
/
ATOMIC_SPECIES
Cl 1.0 Cl.pz-bhs.UPF
ATOMIC_POSITIONS bohr
Cl 0.00 0.00 0.00
Cl 2.00 0.00 0.00
K_POINTS gamma

```

Using `ibrav = 1` we select a simulation box which is simple cubic, with lattice parameter `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 at the Γ point, that is $\mathbf{k} = 0$. This is fine since we want to study a molecule, not an extended crystal. Note that we increased the planewaves cutoff to 100 Ry: this number was obtained from separate convergence tests.

We can now submit a job in order to check that everything will run smoothly, `qsub job.pbs`. In this case it is appropriate to use the submission execution flags `-np 8, -npool 1`.

Incidentally, from the output file of this run (say `c12.out`) 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" c12.out
total energy = -55.44311765 Ry
total energy = -55.73166227 Ry
total energy = -55.83945219 Ry
total energy = -55.84156819 Ry
total energy = -55.84161956 Ry
total energy = -55.84162104 Ry
total energy = -55.84162117 Ry
! total energy = -55.84162119 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-8`.

Now we calculate the total energy as a function of the Cl–Cl bond length. In the reference frame chosen for the input file above we have one Cl atom at (0,0,0) and one at (2,0,0) in units of bohr. 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:

```

sed "s/2.00/NEW/g" c12.in > tmp
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/NEW/${DIST}/g" tmp > c12_${DIST}.in
end

```

If we create this script using copy/paste in a vi window (say vi myscript.tcsh), then we can simply issue tcsh myscript.tcsh in order to generate 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
```

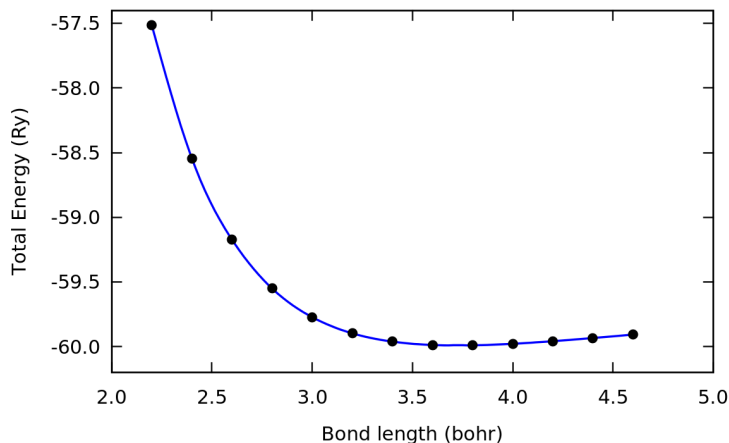
At this point we can execute a batch job which will run pw.x for each of these input files. To this aim we simply duplicate the call to the executable inside our job-1.pbs, and modify this call in order to use the correct input file, eg:

```
mpirun -n 24 pw.x -npool 1 < cl2_2.2.in > cl2_2.2.out
mpirun -n 24 pw.x -npool 1 < cl2_2.4.in > cl2_2.4.out
...
...
mpirun -n 24 pw.x -npool 1 < cl2_4.4.in > cl2_4.4.out
mpirun -n 24 pw.x -npool 1 < cl2_4.6.in > cl2_4.6.out
```

After running these calculations we can look for the total energy as usual:

```
$ grep "\!" cl2_*.out
cl2_2.2.out:!    total energy           =    -57.51390376 Ry
cl2_2.4.out:!    total energy           =    -58.54440037 Ry
cl2_2.6.out:!    total energy           =    -59.17256643 Ry
cl2_2.8.out:!    total energy           =    -59.55021751 Ry
cl2_3.0.out:!    total energy           =    -59.77201221 Ry
cl2_3.2.out:!    total energy           =    -59.89671133 Ry
cl2_3.4.out:!    total energy           =    -59.96077584 Ry
cl2_3.6.out:!    total energy           =    -59.98691109 Ry
cl2_3.8.out:!    total energy           =    -59.98945934 Ry
cl2_4.0.out:!    total energy           =    -59.97764683 Ry
cl2_4.2.out:!    total energy           =    -59.95749243 Ry
cl2_4.4.out:!    total energy           =    -59.93293866 Ry
cl2_4.6.out:!    total energy           =    -59.90658486 Ry
```

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



In this plot the black dots are the calculated datapoints, and the blue 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 $3.725 \text{ bohr} = 1.97 \text{ \AA}$. This value is 1.5% below the measured bond length of 1.99 \AA .

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 $E_{\text{diss}} = E_{\text{Cl}_2} - 2E_{\text{Cl}}$, with E_{Cl} the total energy of an isolated Cl atom.

In order to evaluate this quantity we first calculate E_{Cl_2} using the equilibrium bond length determined in the previous section. For this we modify the input file `c12.in` as follows:

```
...
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

$$E_{\text{Cl}_2} = -59.99059545 \text{ Ry}$$

Now we consider the isolated Cl atom.

The only complication in this case is that the outermost ($3p$) electronic shell of Cl has one unpaired electron: $\uparrow\downarrow \uparrow\downarrow \uparrow$. In order to take this into account we can perform a **spin-polarized** calculation using the following modification of the previous input file:

```
&control
  calculation = 'scf'
  prefix = 'Cl2',
  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
  conv_thr = 1.0d-8
/
```

```
ATOMIC_SPECIES
Cl 1.0 Cl.pz-bhs.UPF
ATOMIC_POSITIONS
Cl 0.00 0.00 0.00
K_POINTS gamma
```

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

$$E_{Cl} = -29.86386108 \text{ Ry}$$

By combining the last two results we find

$$E_{\text{diss}} = 0.262873 \text{ Ry} = 3.58 \text{ eV}$$

This result should be compared to the experimental value of 2.51 eV from https://en.wikipedia.org/wiki/Bond-dissociation_energy. We can see that DFT/LDA overestimates the dissociation energy of Cl_2 by about 1 eV: interatomic bonding is slightly 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 Tutorial 1.2. We can make a new directory, eg

```
cd ~/scratch/summerschool/tutorial-2.1; mkdir silicon
cd silicon
```

and copy the executable, the submission script, the pseudopotential, and the input file from the folder tutorial-1.2. In this case the input file with the converged parameters for planewaves cutoff and Brillouin-zone sampling is:

```
$ more si.in
&control
  calculation = 'scf'
  prefix = 'silicon',
  pseudo_dir = './',
  outdir = './'
/
&system
  ibrav = 2,
  cellldm(1) = 10.28,
  nat = 2,
  ntyp = 1,
  ecutwfc = 25.0,
/
&electrons
  conv_thr = 1.0d-8
/
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
```

The **key observation** in the case of bulk crystals is that often we already have information about the structure from XRD measurements. This information simplifies drastically the calculation of the equilibrium structure.

For example, in the case of silicon, the diamond structure is uniquely determined by the lattice parameter, therefore the energy minimization is a one-dimensional optimization problem, precisely as in the case of the Cl_2 molecule.

In Tutorial 2.2 we will explore the more complicated situation where we want 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 (we can copy/paste this in a vi window: `vi myscript.tcsh` and then execute using `tcsh myscript.tcsh`):

```
sed "s/10.28/NEW/g" si.in > tmp
foreach ALAT ( 10.0 10.1 10.2 10.3 10.4 10.5 10.6 )
sed "s/NEW/${ALAT}/g" tmp > si_${ALAT}.in
end
```

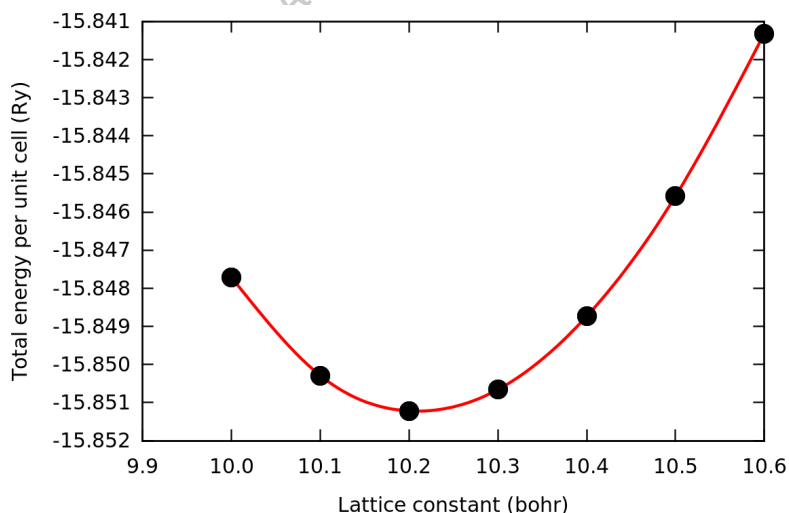
Now we can execute `pw.x` using the generated input files. Once again we can enter all the instances of execution in the same submission script, eg:

```
mpirun -n 12 pw.x -npool 4 < si_10.0.in > si_10.0.out
...
...
mpirun -n 12 pw.x -npool 4 < si_10.6.in > si_10.6.out
```

After running the batch job on the cluster, we should be able to see the output files `si_10.0.out`, ..., `si_10.6.out`, and extract the total energies as follows:

```
$ grep "\!" si_*.out
si_10.0.out:! total energy = -15.84770898 Ry
si_10.1.out:! total energy = -15.85028964 Ry
si_10.2.out:! total energy = -15.85121715 Ry
si_10.3.out:! total energy = -15.85065982 Ry
si_10.4.out:! total energy = -15.84873489 Ry
si_10.5.out:! total energy = -15.84558108 Ry
si_10.6.out:! total energy = -15.84131402 Ry
```

A plot of the total energy vs. lattice parameter is shown below:



Also in this case the black dots are the calculated datapoints, and the red 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 $a = 10.2094$ bohr $= 5.403$ Å. This calculated value is very close to the measured equilibrium parameter of 5.43 Å; DFT/LDA underestimates the measured value 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.

In practice the calculation is 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 one calculation using the same input files as above, this time by setting

```
...
celldm(1) = 10.2094,
...
```

This calculation yields:

$$E_{\text{bulk}} = -15.85122170 \text{ Ry}$$

(this value is an energy per unit cell, and each unit cell contains 2 Si atoms)

For the isolated atom we need to consider one atom per cell, and spin-polarization as in the case of Cl. In fact the outer valence shell of silicon is $2p$: $\uparrow \uparrow \square$.

We can modify the input file as follows (this is very similar to what we have done for the Cl atom, but this time the total spin is 2 Bohr magnetons)

```
&control
  calculation = 'scf'
  prefix = 'silicon',
  pseudo_dir = './',
  outdir = './'
/
&system
 ibrav = 1,
celldm(1) = 20,
nat = 1,
ntyp = 1,
ecutwfc = 25.0,
nspin = 2,
tot_magnetization = 2.0,
occupations = 'smearing',
degauss = 0.001,
/
&electrons
  conv_thr = 1.0d-8
/
```

```
ATOMIC_SPECIES
Si 28.086 Si.pz-vbc.UPF
ATOMIC_POSITIONS
Si 0.00 0.00 0.00
K_POINTS gamma
```

The calculation for the isolated atom gives:

$$E_{\text{Si}} = -7.53189352 \text{ Ry}$$

By combining the last two results we obtain:

$$E_{\text{cohes}} = E_{\text{bulk}}/2 - E_{\text{Si}} = 0.393717 \text{ Ry} = 5.36 \text{ eV}$$

The measured heat of sublimation of silicon is 4.62 eV (see pag. 71 of the Book), therefore DFT/LDA overestimates the experimental value by 16%.

Prof. Feliciano Giustino
University of Oxford
PARADIM School · Cornell, July 2016

An introduction to density functional theory for experimentalists

Tutorial 2.2

Hands-on session

We create a new folder as usual:

```
$ cd ~/scratch/summerschool; mkdir tutorial-2.2 ; cd tutorial-2.2
```

In this hands-on session we will study the equilibrium structure of simple crystals, namely **silicon** (as in Tutorial 2.1), **diamond**, and **graphite**.

Exercise 1

► Familiarize yourself with the steps of Tutorial 2.1, in particular:

- 1 Calculate the equilibrium lattice parameter of silicon
- 2 Calculate the cohesive energy of silicon

Exercise 2

In this exercise we will study the equilibrium structure of diamond.

The crystal structure of diamond is almost identical to the one that we used for silicon in Exercise 1. 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.

► After creating a new directory for this exercise, find a suitable pseudopotential for diamond. It is recommended to use the pseudopotential `C.pz-vbc.UPF`.

The link to the pseudopotential library can be found in the PDF document of Tutorial 1.1.

► Download this pseudopotential, copy over all the necessary files from `tutorial-2.1`, and perform a test run to make sure that everything goes smoothly.

For this test run it is sensible to use the experimental lattice parameter of diamond, 3.56 Å.

► Determine the planewaves kinetic energy cutoff `ecutwfc` required for this pseudopotential.

You can generate the input files for various cutoff energies either manually, or by using the script on pag. 3 of Tutorial 1.2.

You should find that the total energy per atom is converged to within 10 meV when using a cutoff `ecutwfc = 100 Ry`.

► Determine the equilibrium lattice parameter of diamond, by performing calculations similar to those for silicon in Exercise 1.

Compare the calculated lattice parameter with the experimental value.

You should find an equilibrium lattice parameter of 6.66405 bohr = 3.5264 Å.

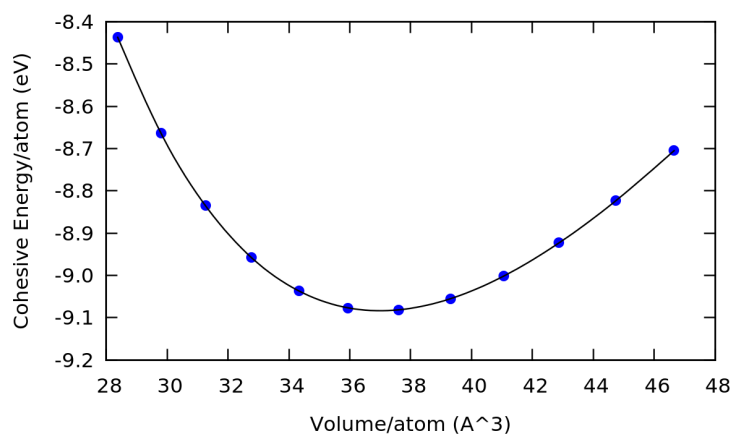
► Using the equilibrium lattice parameter determined in the previous exercise, calculate the cohesive energy of diamond and compare your value with experiments.

For this calculation you can use the same strategy employed in Tutorial 2.1 for the cohesive energy of Si. Note that the C atom in its ground state has a valence electronic configuration $2s \uparrow\downarrow 2p \uparrow \uparrow \square$

As a reference, the cohesive energy that calculated using these settings should be around 9.08 eV (the experimental value is 7.37 eV).

► Plot the cohesive energy vs. volume/atom for all the lattice parameters that you considered.

The plot should look like the following.



Exercise 3

In this exercise we study the equilibrium structure of graphite. A search for carbon allotropes in the Inorganic Crystal Structure Database (ICSD) yields the following structural information:

Summary		Collection Code 76767	
Struct. formula	C	Author	Trucano, P.; Chen, R.
Space Group	P 63/m m c(194)	Title of Article	Structure of graphite by neutron diffraction
Unit Cell	2.464(2) 2.464 6.711(4) 90. 90. 120.	Reference	Nature (London) (1975) 258, p136-p137
Cell Volume	35.29 Å³	Formula Units per Cell	4
Temperature	room temperature	Pressure	atmospheric
PDF-numbers	01-089-7213 41-1487	R-value	0.042
Remark	<input type="text"/> High Quality Data	Warnings & Comments	0 Warnings / 3 Comments

From the data in this page we know that the unit cell of graphite is hexagonal, with lattice vectors

$$\begin{aligned} \mathbf{a}_1 &= a \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \\ \mathbf{a}_2 &= a \begin{pmatrix} -1/2 & \sqrt{3}/2 & 0 \end{pmatrix} \\ \mathbf{a}_3 &= a \begin{pmatrix} 0 & 0 & c/a \end{pmatrix} \end{aligned}$$

($a = 2.464 \text{ \AA}$ and $c/a = 2.724$), and with 4 C atoms per primitive unit cell, with fractional coordinates:

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

► Starting from the input file that you used for diamond in Exercise 2, 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 `cellldm()` in the input. Search for these entries in the documentation page:

http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PW.html

Here you should find the following:

Namelist: SYSTEM

ibrav	INTEGER
Status:	REQUIRED

Bravais-lattice index. If `ibrav` $\neq 0$, specify EITHER [`cellldm(1)`-`cellldm(6)`] OR [`A,B,C,cosAB,cosAC,cosBC`] but NOT both. The lattice parameter "alat" is set to `alat = cellldm(1)` (in a.u.) or `alat = A` (in Angstrom); see below for the other parameters.
For `ibrav=0` specify the lattice vectors in `CELL_PARAMETER`, optionally the lattice parameter `alat = cellldm(1)` (in a.u.) or `= A` (in Angstrom), or else it is taken from `CELL_PARAMETERS`

ibrav	structure	cellldm(2)-cellldm(6) or: b,c,cosab,cosac,cosbc
0	free	crystal axis provided in input: see card <code>CELL_PARAMETERS</code>
1	cubic P (sc)	$v_1 = a(1,0,0), v_2 = a(0,1,0), v_3 = a(0,0,1)$
2	cubic F (fcc)	$v_1 = (a/2)(-1,0,1), v_2 = (a/2)(0,1,1), v_3 = (a/2)(-1,1,0)$
3	cubic I (bcc)	$v_1 = (a/2)(1,1,1), v_2 = (a/2)(-1,1,1), v_3 = (a/2)(-1,-1,1)$
4	Hexagonal and Trigonal P	<code>cellldm(3)=c/a</code> $v_1 = a(1,0,0), v_2 = a(-1/2,\sqrt{3}/2,0), v_3 = a(0,0,c/a)$

Based on this information we must use `ibrav = 4` and `cellldm(1)` and `cellldm(3)`.

As a sanity check, if you run a calculation with `ecutwfc = 100` and `K_POINTS gamma` you should obtain a total energy of -44.581847 Ry.

► A convergence test with respect to the number of **k**-points indicates that the total energy is converged to 4 meV/atom when using a shifted $6 \times 6 \times 2$ grid (6 6 2 1 1 1 with `K_POINTS automatic`). Using this setup for the Brillouin-zone sampling, calculate the lattice parameters of graphite *a* and *c/a* at equilibrium. Note that this will require 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 `graph.in`:

- Replace the values of `cellldm(1)` and `cellldm(3)` by the placeholders `ALAT` and `RATIO`, respectively;
- Create a script `myscript.tcsh` with the following content:


```

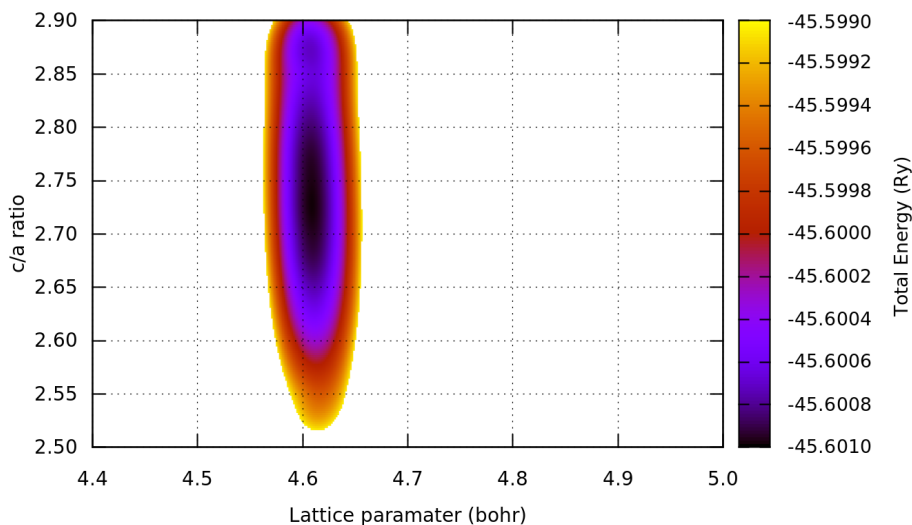
rm tmp.pbs
foreach A ( 4.4 4.5 4.6 4.7 4.8 4.9 5.0 )
  foreach CA ( 2.50 2.55 2.60 2.65 2.70 2.75 2.80 2.85 2.90 )
    sed "s/ALAT/${A}/g" graph.in > tmp
    sed "s/RATIO/${CA}/g" tmp > graph_${A}_${CA}.in
    echo "mpirun -n 12 pw.x -npool 12 < graph_${A}_${CA}.in > graph_${A}_${CA}.out" >> tmp.pbs
  end
end

```

- By running `tcsh myscript.tcsh` you will be able to generate input files for all these combinations of a and c/a .
The file `tmp.pbs` will contain all the correct execution commands, that you can copy/paste directly inside your submission script.
- Note that this will produce $7 \times 9 = 63$ input files, but the total execution time on 12 cores should be around 1 min.
- At the end you will be able to extract the total energies by using `grep` as usual

```
grep "\!" graph_*.out > mydata.txt
```

If you plot the total energies that you obtained as a function of a and c/a you should be able to get something like the following:



This plot was generated using the following commands in `gnuplot` (the file `mydata.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] "mydata.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.

By zooming in a plot like the one above you should be able to find the following equilibrium lattice parameters:

$$a = 2.439 \text{ \AA}, c/a = 2.729$$

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 cannot correctly predict the interlayer binding in graphite due to the lack of van der Waals corrections. Since LDA generally tends to overbind (as we have seen in all examples studied so far), but it does not contain van der Waals corrections, this functional works well for graphite owing to a cancellation of errors.

For future reference let us just note that the total energy calculated using these optimized lattice parameters is -45.60104552 Ry.

Exercise 4

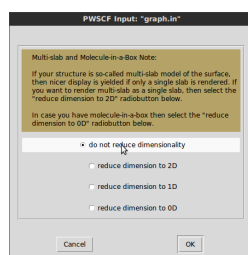
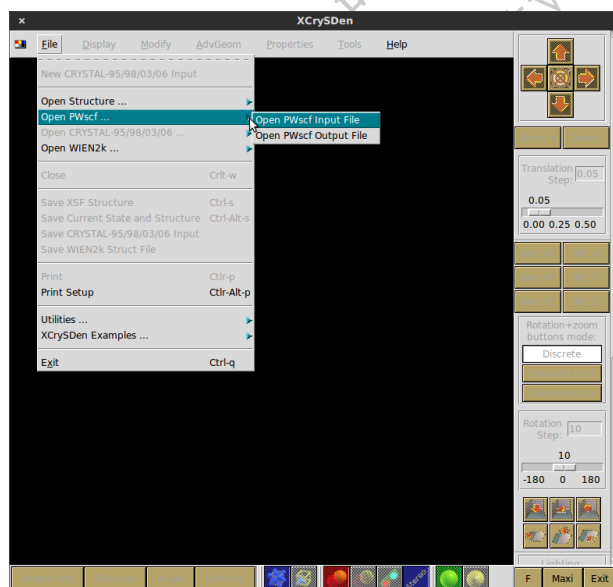
In this exercise we want to see how the structure of graphite that we are using in our input file looks like in a ball-and-stick model.

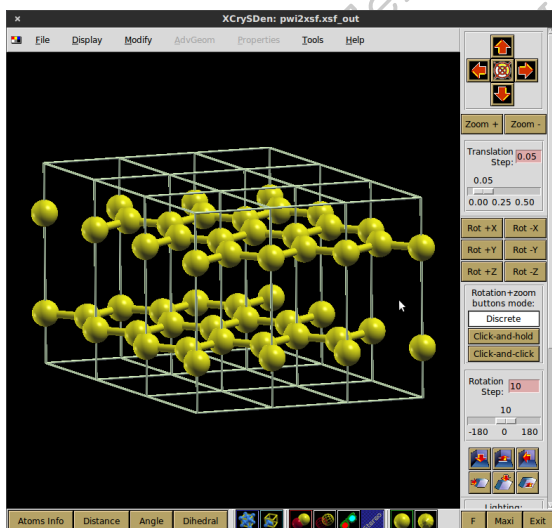
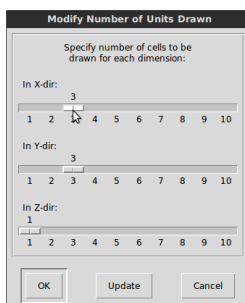
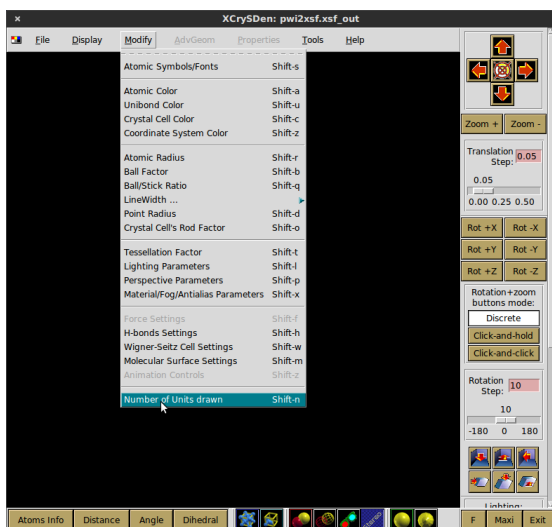
The software [xcryden](http://www.xcryden.org) can import QE input files and visualize the atomistic structures. General info about this project can be found at <http://www.xcryden.org>.

We launch xcryden by typing:

```
$ xcryden
```

The user interface is very simple and intuitive. The following snapshots may be helpful to get started with the visualization.





Exercise 5

- ▶ Which carbon allotrope is more stable at ambient conditions, diamond or graphite?

Note: The answer to this question is rather delicate. In nature graphite is more stable than diamond by 40 meV/atom at ambient pressure and low temperature.

Using the calculations of Exercises 2 and 3 we find that the cohesive energy of diamond is 8 meV lower than in graphite. Therefore DFT/LDA would predict diamond to be more stable, contrary to experiments. This is in agreement with the following study by Janotti et al, <http://dx.doi.org/10.1103/PhysRevB.64.174107>.

An introduction to density functional theory for experimentalists

Tutorial 3.1

We begin with a new folder as usual:

```
$ cd ~/scratch/summerschool ; mkdir tutorial-3.1 ; cd tutorial-3.1
```

In the first part of this tutorial we will say more on the determination of equilibrium structures of molecules and solids. In the second part we will attempt a calculation of elastic properties.

Automatic optimization of atomic coordinates

In Tutorial 2.1 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 Cl₂.

As a starter we copy over the input files that we used for the exercise on Cl₂ during Tutorial 2.1:

```
$ cp ../tutorial-2.1/cl2.in ./
$ cp ../tutorial-2.1/pw.x ./
$ cp ../tutorial-2.1/job.pbs ./
```

We check that everything is in place and that `job.pbs` is still reading in correctly `cl2.in` (in case this was modified during Tutorial 2.1). As usual we perform a test run in order to make sure that everything goes smoothly:

```
$ qsub job.pbs
```

For this job we must remember to use the submission flag `-npool 1`, otherwise the code will crash (can you explain why?). If everything is still fine, 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:

http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PW.html

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

Namelist: CONTROL

calculation	CHARACTER
<i>Default:</i>	'scf'
a string describing the task to be performed:	
'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 `c12.in`:

```
&control
  calculation = 'relax'
  prefix = 'C12',
  ...
```

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 certain threshold, and the total potential energy surface has changed less than a given threshold from the previous iteration.

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

<code>forc_conv_thr</code>	REAL
<i>Default:</i>	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	

[\[Back to Top\]](#)

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

<code>etot_conv_thr</code>	REAL
<i>Default:</i>	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	

[\[Back to Top\]](#)

In principle we could perform calculations without specifying these parameters; in this case `pw.x` will use some preset default values. For the sake of completeness let us specify some rather stringent criterion in the input file `c12.in`:

```
&control
  calculation = 'relax'
  prefix = 'C12',
  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
  conv_thr = 1.0d-8
/
&ions
/
ATOMIC_SPECIES
..

```

This extra card is to specify runtime parameters, but we can leave it 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 `c12.out` directly using `vi`. As a reminder, 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 (Ry/au):

atom   1 type   1   force =    0.12392756    0.00000000    0.00000000
atom   2 type   1   force =   -0.12392756    0.00000000    0.00000000

Total force =    0.175260    Total SCF correction =    0.000022

```

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      0.123927558    0.000000000    0.000000000
Cl      4.876072442    0.000000000    0.000000000

```

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 (Ry/au):

atom   1 type   1   force =    0.00000147    0.00000000    0.00000000
atom   2 type   1   force =   -0.00000147    0.00000000    0.00000000

Total force =    0.000002    Total SCF correction =    0.000024
...
Final energy   =   -59.9905957153 Ry
Begin final coordinates

```

```

ATOMIC_POSITIONS (bohr)
Cl      0.637701785  0.000000000  0.000000000
Cl      4.362298215  0.000000000  0.000000000
End final coordinates

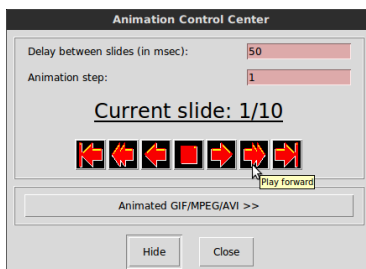
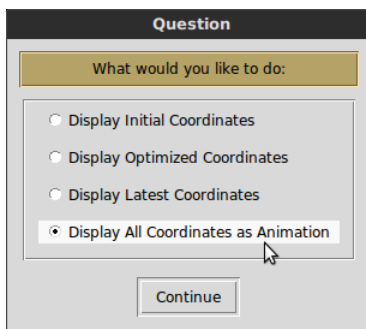
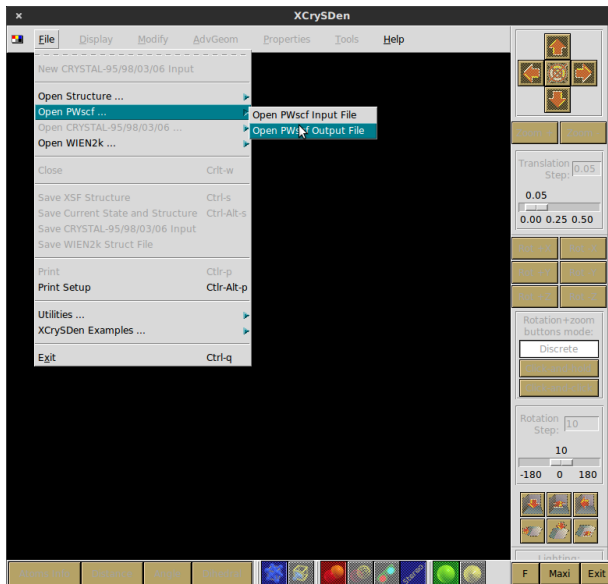
```

Here we see that the forces are essentially vanishing, therefore we reached the equilibrium configuration. The total energy at equilibrium is -59.9905957153 Ry, and the bond length is 3.7246 bohr = 1.971 Å. These values are in agreement with what we had found in Tutorial 2.1 (pag. 4) 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" c12.out
```

There is also a way to directly visualize 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
<i>Default:</i> 'scf'	
a string describing the task to be performed: 'scf', 'nscf', 'bands', 'relax', 'md', 'vc-relax', 'vc-md'	
(vc = variable-cell).	

Let us consider the case of **graphite** as in Tutorial 2.2. We can copy over the corresponding input file and the carbon pseudopotential:

```
$ cp ../tutorial-2.2/exercise3/graph.in ./
$ cp ../tutorial-2.2/exercise3/C.pz-vbc.UPF ./
```

Let us modify the input file in 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
  conv_thr = 1.0d-8
/
&ions
/
&cell
/
ATOMIC_SPECIES
  C 1.0 C.pz-vbc.UPF
ATOMIC_POSITIONS crystal
  C 0.00 0.00 0.25
  C 0.00 0.00 0.75
  C 0.333333 0.666666 0.25
  C 0.666666 0.333333 0.75
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 calculation. 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 so as to reduce the stress.

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:

```
/ subroutine stress
```

We will see something like:

```
entering subroutine stress ...

      total   stress (Ry/bohr**3)                (kbar)      P= 4044.22
0.03445761  0.00000000  0.00000000      5068.89   0.00   0.00
0.00000000  0.03445761  0.00000000           0.00 5068.89   0.00
0.00000000  0.00000000  0.01356098           0.00   0.00 1994.89
```

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

We can note that in this case the individual forces on the atoms are all vanishing by symmetry:

```
Forces acting on atoms (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 = 200.62641 a.u.^3 ( 29.72977 Ang^3 )

CELL_PARAMETERS (alat= 4.00000000)
  1.149452048  0.000000000  0.000000000
 -0.574726024  0.995454674  0.000000000
  0.000000000  0.000000000  2.739654388

ATOMIC_POSITIONS (crystal)
C      0.000000000  0.000000000  0.250000000
C      0.000000000  0.000000000  0.750000000
C      0.333333000  0.666666000  0.250000000
C      0.666666000  0.333333000  0.750000000
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 $a_{\text{lat}} = 4.0$ bohr. Therefore the optimized lattice parameter is now

$$a = 4.00000000 \cdot 1.149452048 = 4.59781 \text{ bohr} = 2.433 \text{ \AA}$$

The optimized c/a ratio is

$$c/a = 2.739654388/1.149452048 = 2.383.$$

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 -45.59332176 Ry.

Note. From this calculation we have obtained a c/a ratio which is much smaller than the one determined in Tutorial 2.2 by studying the potential energy surface (2.383 here vs. 2.729 in T2.2). The interlayer separation is approximately 15% shorter in the present calculation. This result is a calculation artifact. What is happening is that the code modifies the 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. In order to avoid this problem, `pw.x` performs one additional calculation at the very end, after having redefined all the \mathbf{G} -vectors according to the optimized structure. We can see that this step yields a residual pressure of 92.7 kbar along the c -axis. 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.

Elastic constants of diamond

Now we want study the elastic constants of diamond. As we have seen in Lecture 3.2, 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 the lecture. In particular:

- We consider an **isotropic** deformation of the diamond structure. This corresponds to a uniform stretch of the lattice vectors:

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

The resulting change in the total potential energy from the equilibrium value U_0 is:

$$U - U_0 = \Omega \frac{3}{2} (C_{11} + 2C_{12}) \eta^2 \quad (1)$$

The calculation of U and U_0 can be performed by using the following input file for diamond (taken from Tutorial 2.2, Exercise 2):

```

$ cat > scf.in << EOF
&control
  calculation = 'scf'
  prefix = 'diamond',
  pseudo_dir = './',
  outdir = './'
/
&system
 ibrav = 0,
  celldm(1) = 6.66405,
  nat = 2,
  ntyp = 1,
  ecutwfc = 100.0,
/
&electrons
  conv_thr = 1.0d-8
/
ATOMIC_SPECIES
  C 1.0 C.pz-vbc.UPF
ATOMIC_POSITIONS
  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
  -0.50 0.00 0.50
   0.00 0.50 0.50
  -0.50 0.50 0.00
EOF

```

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 keyword `CELL_PARAMETERS`.

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

$$U_0 = -22.80164823 \text{ Ry.}$$

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

This gives $\Omega = 73.9869 \text{ bohr}^3$.

Now we can modify this input file in order to set in an isotropic deformation with $\eta = 0.002$:

```

$ more scf_iso.in
...
CELL_PARAMETERS
  -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:

$$U = -22.80159963 \text{ Ry}$$

Using Eq. (1) with $\eta = 0.002$ we obtain

$$C_{11} + 2C_{12} = 0.10948 \text{ Ry/bohr}^3 = 1610.5 \text{ GPa} \qquad 1 \text{ Ry/bohr}^3 = 14710.5 \text{ GPa}$$

- We now consider a **tetragonal** deformation.

From Lecture 3.2 we have the relation:

$$U - U_0 = \Omega 3 (C_{11} - C_{12}) \eta^2 \qquad (2)$$

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 modify the input file as follows:

```
$ more scf_tetra.in
```

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

This calculation gives:

$$U = -22.80156418 \text{ Ry}$$

Using Eq. (2) with $\eta = 0.002$ we obtain

$$C_{11} - C_{12} = 0.094668 \text{ Ry/bohr}^3 = 1392.6 \text{ GPa}$$

By combining the two relations for C_{11} and C_{12} we obtain:

$$C_{11} = 1465.6 \text{ GPa}, C_{12} = 72.6 \text{ GPa}.$$

The corresponding experimental values are $C_{11} = 1079 \text{ GPa}$, $C_{12} = 124 \text{ GPa}$, from [McSkimin & Andreatch, J. Appl. Phys. 43, 2944 \(1972\)](#).

Note. These calculations are not fully converged, and by refining our setup we can obtain better agreement with experiment. In particular, we are determining elastic constants using only 2 calculations in each case. Since elastic constants are second derivatives of the total energy, a much more accurate approach is to evaluate such derivatives using 3 total energy calculations. See Exercise 6.4 of the Book for how to perform more refined calculations.

- Finally we can consider a **trigonal** deformation.

From Lecture 3.2 we have the relation:

$$U - U_0 = \Omega \frac{1}{2} C_{44} \eta^2 \quad (3)$$

and the trigonal distortion of the unit cell can be realized by considering the following distortion for $\eta = 0.002$:

```
$ more scf_trigo.in
```

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

This calculation gives:

```
U = -22.80164338 Ry
```

Using Eq. (3) with $\eta = 0.002$ we obtain

$$C_{44} = 0.0327761 \text{ Ry/bohr}^3 = 482.2 \text{ GPa}$$

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

Note. These calculations are not fully converged: in order to obtain accurate results we need to use higher-order finite difference formulas. See Exercise 6.4 of the Book.

An introduction to density functional theory for experimentalists

Tutorial 3.2

Hands-on session

We create a new folder as usual:

```
$ cd ~/scratch/summerschool; mkdir tutorial-3.2 ; cd tutorial-3.2
```

In this hands-on session we will first familiarize ourselves with calculations of elastic constants, using **diamond** as a test case. Then we will try to set up an entirely new calculation on **SrTiO₃**; here we will use the Materials Project database to find the initial geometry.

Exercise 1

► Calculate the elastic constants C_{11} , C_{12} , and C_{44} of diamond, by following the steps illustrated in Tutorial 3.1.

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 experiment.

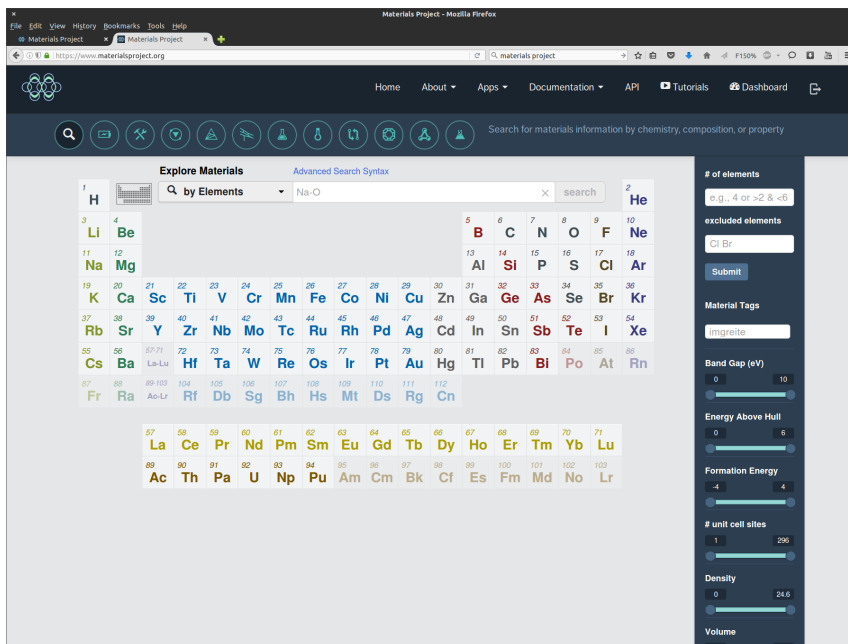
► In Tutorial 3.1 we used a 'deformation' parameter $\eta = 0.002$ in all our calculations. Investigate the sensitivity of the calculated bulk modulus of diamond to the choice of η , by repeating the calculations for $\eta = 0.1$, 0.01 , and 0.001 .

Exercise 2

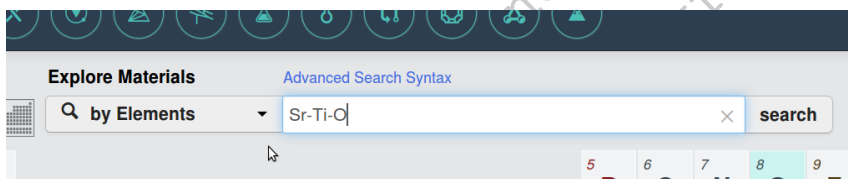
In this exercise we want to set up a simple input file to study SrTiO₃. In order to find an initial guess for the unit cell and atomic coordinates, we search the Materials Project database.

If you do not already have an account on the Materials Project, 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.

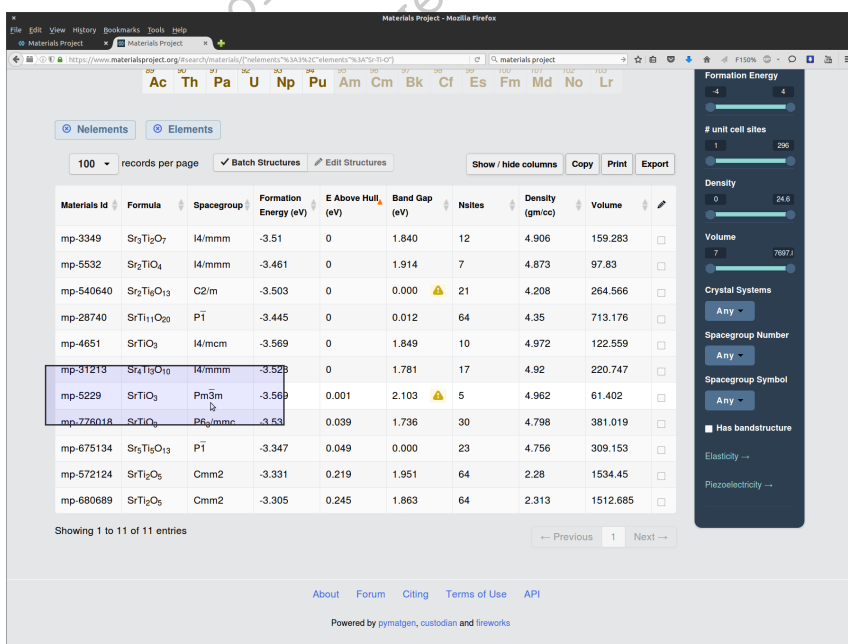
After logging-in you should be able to see a periodic table like the following:



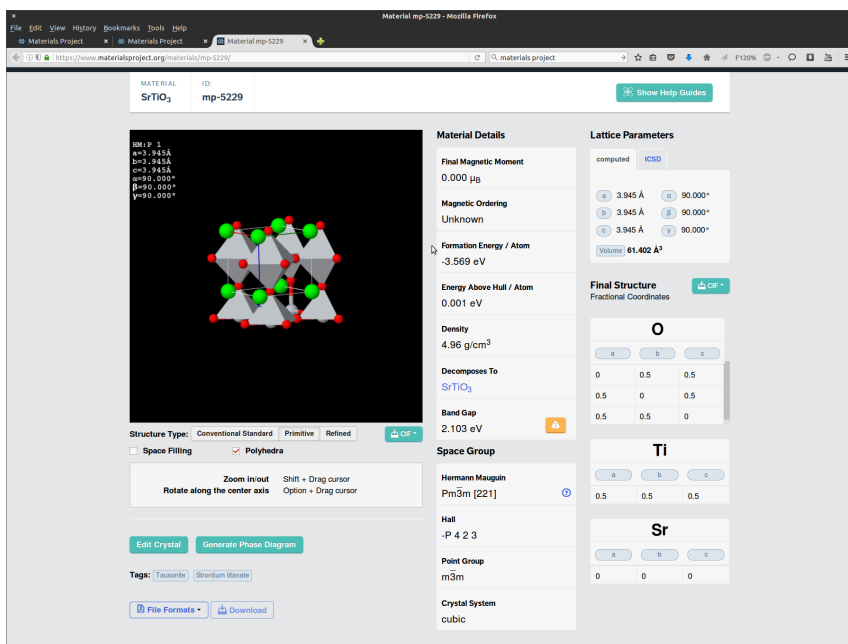
Now we can search for SrTiO_3 by entering Sr, Ti, and O in the search bar:



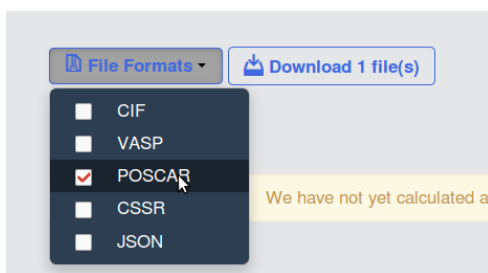
We are looking for the $Pm\bar{3}m$ structure of SrTiO_3 , which is the high-temperature cubic phase. We will study the cubic phase since it only contains 5 atoms, therefore calculations are relatively easy.



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 DFT package.

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

```

1  Sr1 Ti1 O3
2  1.0
3  3.945130 0.000000 0.000000
4  0.000000 3.945130 0.000000
5  0.000000 0.000000 3.945130
6  Sr Ti O
7  1 1 3
8  direct
9  0.000000 0.000000 0.000000 Sr
10 0.500000 0.500000 0.500000 Ti
11 0.500000 0.000000 0.500000 O
12 0.500000 0.500000 0.000000 O
13 0.000000 0.500000 0.500000 O

```

Here the first line is a comment field, the second line contains the lattice parameter a in \AA . 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 $a = 1$ and to give the lattice vectors directly in \AA). 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 states 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 SrTiO₃ using pw.x. For the time being we can leave the pseudopotential field blank. It is absolutely fine to start from an input file that you have from previous exercises.

If anything is unclear, please consult the documentation at

http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PW.html

Note: Remember that `celldm(1)` is to be given in atomic units. In order 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, Goedecker & Hutter.

► Download the required pseudopotential files from the QE library, using `wget` as in Tutorial 1.1.

► Using the input file just created, say `scf.in`, perform a test run using `calculation = 'scf'` in order 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`.

Exercise 3

Now that we have a basic setup for SrTiO₃, we need to perform convergence tests.

► Determine the planewaves kinetic energy cutoff which is required to have the total energy converged to within 50 meV/atom. For this calculation you can use the same `K_POINTS` set of Exercise 2. As a reminder, we performed a similar operation in Tutorial 1.2/Exercise 2.

► Using the cutoff just obtained, determine the sampling of the Brillouin zone required to have the total energy converged to within 10 meV/atom.

You can find an example of such a test in Tutorial 1.2/Exercise 3.

Exercise 4

► Using the convergence parameters obtained in Exercise 3, determine the optimized lattice parameter of SrTiO₃ by using a calculation of type `vc-relax` (see Tutorial 3.1 for an example).

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.

► Compare your optimized lattice parameter with the experimental value from [Cao et al, PSSA 181, 387 \(2000\)](#).

Exercise 5

As a sanity check, at the end of Exercise 3 and 4 we should have obtained the following parameters:

```
...
celldm(1) = 7.18899,
ecutfwc = 210,
...
K_POINTS
4 4 4 1 1 1
```

► Use these parameters to calculate the bulk modulus of cubic SrTiO₃.

Note: In order to obtain reasonably accurate results it is convenient to use Eq. (1) of Tutorial 3.1, after rewriting as follows:

$$B = \frac{1}{3}(C_{11} + 2C_{12}) = \frac{1}{9} \frac{1}{\Omega} \frac{\partial^2 U}{\partial \eta^2} \simeq \frac{1}{9} \frac{1}{\Omega} \frac{U(+\eta) - 2U(0) + U(-\eta)}{\eta^2}$$

This expression shows that we can calculate the bulk modulus by using the second derivative of the total energy with respect to the deformation parameter. The second derivative is then approximated using a finite-difference formula involving 3 points (+ η , 0, - η).

For your reference, using $\eta = 0.01$ and considering the unit cell volume $\Omega = 360.5033 \text{ bohr}^3$, the value $B = 189 \text{ GPa}$ is obtained.

► Following the same lines as in the last step, calculate the elastic constants C_{11} and C_{12} of cubic SrTiO₃.

Hint: We already have the bulk modulus, therefore we know $C_{11} + 2C_{12} = 3B$. What we still need is the difference $C_{11} - C_{12}$, as discussed in Tutorial 3.1.

Also in this case we can rewrite Eq. (2) of Tutorial 3.1 as:

$$C_{11} - C_{12} \simeq \frac{1}{6} \frac{1}{\Omega} \frac{U(+\eta) - 2U(0) + U(-\eta)}{\eta^2}$$

and we perform three calculations corresponding to a tetragonal deformation of the lattice.

As a reference, you should obtain values in the range of $C_{11} = 360 \text{ GPa}$ and $C_{12} = 104 \text{ GPa}$.

► Compare your calculated constants B , C_{11} , and C_{12} with the experimental values of [Bell & Rupprecht, Phys. Rev. 129, 90 \(1963\)](#).

You should find that the deviation from experiment is smaller than 10%.

► Can you think of possible strategies to improve your results?

An introduction to density functional theory for experimentalists

Tutorial 4.1

We create a new folder:

```
$ cd ~/scratch/summerschool ; mkdir tutorial-4.1 ; cd tutorial-4.1
```

In this tutorial we will learn how to calculate the vibrational frequencies of molecules and solids, phonon dispersion relations, LO-TO splitting, IR activity, and low-frequency dielectric constants.

Stretching frequency of a diatomic molecule

We start from the simplest possible system, the diatomic molecule Cl_2 studied in Tutorial 2.1.

We copy the setup from T2.1:

```
$ cp ../tutorial-2.1/cl2.in ./
$ cp ../tutorial-2.1/Cl.pz-bhs.UPF ./
$ cp ../tutorial-2.1/pw.x ./
```

We now modify the input file in order to make sure that we are using the optimized geometry and convergence parameters from Tutorial 2.1:

```
$ 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
  conv_thr = 1.0d-8
/
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. In this case it is important to set `-npool 1` in the call to `pw.x`.

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.

In the Quantum Espresso package DFPT for vibrations is implemented in a code named `ph.x`. In order to use this code we need to go back to the root directory `summerschool/espresso-5.4.0`, and execute:

```
$ make ph
$ cp bin/ph.x ../tutorial-4.1/
```

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

```
$ cat > 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; the file 'cl2.dyn' will contain the dynamical matrix. `amass` is the atomic mass in amu (atomic mass units). 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. Note that `prefix` must be the same as that used by `pw.x`.

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:

```
$ more cl2.in

...
K_POINTS tpiba
1
0.0 0.0 0.0 1.0
```

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 previous version 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 insert the following lines into our submission script, and run the job:

```
mpirun -np 12 pw.x -npool 1 < cl2.in > cl2.out
mpirun -np 12 ph.x -npool 1 < ph.in > ph.out
```

After completion of this job we should find the file `cl2.dyn` in our working directory:

```
$ more cl2.dyn
```

Dynamical matrix file

```
1 2 1 20.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
1 1 'Cl ' 32310.698383589064
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
0.39656662 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00346652 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00000000 0.00000000 0.00346652 0.00000000

1 2
-0.42782957 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00482745 0.00000000 -0.00000000 0.00000000
0.00000000 0.00000000 -0.00000000 0.00000000 0.00482745 0.00000000

2 1
-0.42783017 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00482799 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00000000 0.00000000 0.00482799 0.00000000

2 2
0.39662970 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00352411 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00000000 0.00000000 0.00352411 0.00000000
```

Diagonalizing the dynamical matrix

```
q = ( 0.000000000 0.000000000 0.000000000 )

*****
freq ( 1) = -3.234445 [THz] = -107.889470 [cm-1]
( -0.707133 -0.000000 -0.000003 -0.000000 -0.000055 -0.000000 )
( -0.707081 -0.000000 -0.000003 -0.000000 0.000028 -0.000000 )
freq ( 2) = -0.668245 [THz] = -22.290250 [cm-1]
( 0.000015 0.000000 -0.662456 0.000000 -0.253313 0.000000 )
( 0.000015 0.000000 0.658488 0.000000 0.251752 0.000000 )
freq ( 3) = -0.667934 [THz] = -22.279866 [cm-1]
( -0.000039 0.000000 -0.253258 0.000000 0.662427 0.000000 )
( -0.000039 0.000000 0.251807 0.000000 -0.658516 0.000000 )
freq ( 4) = 1.669664 [THz] = 55.694000 [cm-1]
( -0.000007 0.000000 0.656044 0.000000 0.258052 0.000000 )
( -0.000008 0.000000 0.659998 0.000000 0.259651 0.000000 )
freq ( 5) = 1.669785 [THz] = 55.698035 [cm-1]
( 0.000009 0.000000 0.258107 0.000000 -0.656073 0.000000 )
( 0.000013 0.000000 0.259596 0.000000 -0.659969 0.000000 )
freq ( 6) = 16.617980 [THz] = 554.316153 [cm-1]
( -0.707081 0.000000 0.000000 0.000000 0.000002 0.000000 )
( 0.707133 0.000000 -0.000000 0.000000 0.000002 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.

Here we see that some frequencies are negative. This is only a convention, which is used 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 codes you may find the imaginary unit i in front of these frequencies, eg 107.889470 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. However, we should keep in mind 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 space is not exactly 'isotropic', owing to our 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. This procedure corresponds to modifying the dynamical matrix in such a way as to make sure that the molecule will not experience any restoring force when translated or rotated. We can perform this operation by calling a **post-processing** program, `dynmat.x`:

```
$ cp ../espresso-5.4.0/bin/dynmat.x ./
$ cat > c12.dynmat.in << EOF
&input
  fildyn = 'c12.dyn',
  asr = 'zero-dim',
/
EOF
$ ./dynmat.x < c12.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 submitting a batch job.

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	554.32	16.6180	0.0000

We see that now the system has only one nonzero vibrational frequency, as expected. The calculated value 68.7 meV ($1 \text{ meV} = 8.0655 \text{ cm}^{-1}$) is close to our result from the previous section, 65.1 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 $\delta \rightarrow 0$.

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

An extensive set of examples on how to use `ph.x` is located inside the the directory:
[espresso-5.4.0/PHonon/examples/](http://www.quantum-espresso.org/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 Tutorial 2.2:

```
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/C.pz-vbc.UPF
$ cat > scf.in << EOF
&control
  calculation = 'scf'
  prefix = 'diamond',
  pseudo_dir = './',
  outdir = './'
/
&system
 ibrav = 2,
  celldm(1) = 6.66405,
  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
EOF
```

Here the lattice constant, the Brillouin-zone sampling, and the planewaves cutoff are set to the same values that we obtained in Tutorial 2.2. We are now using a more stringent threshold for the self-consistent cycle, `conv_thr`, 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 corresponding interatomic force constants in real space;
- 3) Calculate the frequencies along the chosen path of \mathbf{q} -points, using a Fourier interpolation.

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

```
$ cat > ph.in << EOF
-
&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 using the following lines in our job submission script:

```
mpirun -np 12 pw.x -npool 4 < scf.in > scf.out
mpirun -np 12 ph.x -npool 4 < ph.in > 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 `../espresso-5.4.0/bin`. The input file is very simple, and we can execute this program locally (ie without submitting to the queue):

```
$ cp ../espresso-5.4.0/bin/q2r.x ./
$ cat > q2r.in << EOF
&input
  fildyn = 'dyn',
  flfrc = 'diam.fc'
/
EOF
$ ./q2r.x < 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 in `../espresso-5.4.0/bin`.

```
$ cp ../espresso-5.4.0/bin/matdyn.x ./
```

The input file is on the next page, and also this program can be executed locally:

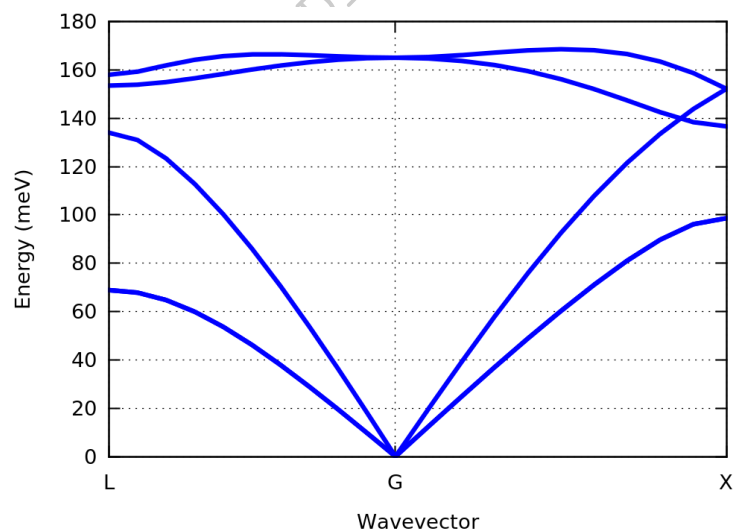
```

$ cat > matdyn.in << EOF
&input
  asr = 'simple',
  flfrc = 'diam.fc',
  flfrq = 'diam.freq'
/
21
0.500 0.500 0.500
0.450 0.450 0.450
0.400 0.400 0.400
0.350 0.350 0.350
0.300 0.300 0.300
0.250 0.250 0.250
0.200 0.200 0.200
0.150 0.150 0.150
0.100 0.100 0.100
0.050 0.050 0.050
0.000 0.000 0.000
0.100 0.000 0.000
0.200 0.000 0.000
0.300 0.000 0.000
0.400 0.000 0.000
0.500 0.000 0.000
0.600 0.000 0.000
0.700 0.000 0.000
0.800 0.000 0.000
0.900 0.000 0.000
1.000 0.000 0.000
EOF
$ ./matdyn.x < matdyn.in

```

In this file we are specifying that we want the code to calculate vibrational frequencies for 21 \mathbf{q} -points. The Cartesian coordinates of these points are specified in units of $2\pi/a$. In this example we have 21 points uniformly distributed along 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` gives the following phonon dispersion relations:



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

In this section 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: i) The vibrational frequencies of longitudinal and transverse optical phonons at long wavelength ($\mathbf{q} \rightarrow 0$) do not coincide. This is called LO-TO splitting. ii) The system exhibits infrared (IR) activity. iii) 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:

```
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/Ga.pz-bhs.UPF
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/As.pz-bhs.UPF
cat > scf.in << EOF
&control
  calculation = 'scf'
  prefix = 'gaas',
  pseudo_dir = './',
  outdir = './'
/
&system
 ibrav = 2,
  celldm(1) = 10.4749,
  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
```

All the parameters in this input file have been optimized separately. We can perform a test run to make sure that everything is in place: as usual we call `pw.x` from within our submission script:

```
mpirun -np 12 pw.x -npool 12 < scf.in > scf.out
```

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 section. The only differences are the two additional flags `epsil` and `zeu`:

```
$ cat > 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,
/
0.0 0.0 0.0
EOF
```

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

[\[Back to Top\]](#)

epsil	LOGICAL
<i>Default:</i> .false.	
<p>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.</p>	

[\[Back to Top\]](#)

zeu	LOGICAL
<i>Default:</i> zeu=epsil	
<p>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.</p>	

Therefore these flags instruct `ph.x` to also evaluate the high-frequency (electronic) dielectric constant tensor of the system, as well as the Born effective charges. As we have seen in Lecture 4.2, 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:

```
mpirun -np 12 ph.x -npool 12 < ph.in > ph.out
```

Towards the end of the output file we will 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

( 11.559429679  0.000000000  0.000000000 )
( 0.000000000  11.559429679  0.000000000 )
( 0.000000000  0.000000000  11.559429679 )

Effective charges (d Force / dE) in cartesian axis

atom 1 Ga
Ex ( 2.03189  0.00000  0.00000 )
Ey ( 0.00000  2.03189  0.00000 )
Ez ( 0.00000  0.00000  2.03189 )
atom 2 As
Ex ( -2.04609  0.00000  0.00000 )
Ey ( 0.00000 -2.04609  0.00000 )
Ez ( 0.00000  0.00000 -2.04609 )

Diagonalizing the dynamical matrix

q = ( 0.000000000  0.000000000  0.000000000 )

*****
freq ( 1) = 0.142309 [THz] = 4.746905 [cm-1]
freq ( 2) = 0.142309 [THz] = 4.746905 [cm-1]
freq ( 3) = 0.142309 [THz] = 4.746905 [cm-1]
freq ( 4) = 8.264350 [THz] = 275.669027 [cm-1]
freq ( 5) = 8.264350 [THz] = 275.669027 [cm-1]
freq ( 6) = 8.264350 [THz] = 275.669027 [cm-1]
*****
```

Here we recognize the high-frequency dielectric constant of GaAs, $\epsilon_\infty = 11.56$, and the Born effective charges of Ga and As, respectively $Z_{\text{Ga}}^* = 2.03$ and $Z_{\text{As}}^* = -2.04$ (in principle these two values should add up to zero, but we have some numerical error).

The calculated dielectric constant is about 6% higher than the experimental value, $\epsilon_\infty^{\text{exp}} = 10.89$. This overestimation is related to the band gap problem of DFT, which will be discussed in Lecture 5.1.

In the previous page we can see that the 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 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 > 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 $\mathbf{q} \rightarrow 0$ (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 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     275.63   8.2632   2.6559
   5     275.63   8.2632   2.6559
   6     295.77   8.8670   2.6559

Electronic dielectric permittivity tensor (F/m units)
  11.559430   0.000000   0.000000
   0.000000  11.559430   0.000000
   0.000000   0.000000  11.559430

... with zone-center polar mode contributions
  13.080194  -0.000000   0.000000
   0.000000  13.310576   0.000000
   0.000000   0.000000  13.310576
```

We can see that now we have 2 degenerate TO modes at 34.17 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 $\epsilon_0 = 13.08$. For comparison the experimental value is $\epsilon_0^{\text{exp}} = 12.9$, therefore the relative deviation is of only 1.4%.

In the output file we also see the IR activities in units of $\text{debye}/\text{\AA}^2/\text{amu}$.

We should be careful in interpreting these data: IR 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. The procedure that we used to calculate 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 simply use:

```
phonons of GaAs near Gamma
&inputph
  prefix = 'gaas',
  amass(1) = 69.723,
  amass(2) = 74.9216,
  fildyn = 'dyn',
  tr2_ph = 1.0d-14,
/
0.01 0.0 0.0
```

This gives two degenerate TO phonons at 34.17 meV and one LO phonon at 36.67 meV, in agreement with our previous calculation.

This alternative procedure is perfectly legitimate, but it does not provide us with Born charges, dielectric constants, and IR activities.

An introduction to density functional theory for experimentalists

Tutorial 4.2

Hands-on session

We create a new folder as usual:

```
> cd ~/scratch/summerschool; mkdir tutorial-4.2 ; cd tutorial-4.2
```

In this tutorial we study the phonon dispersion relations of **diamond**, **GaAs** and **SrTiO₃**.

Exercise 1

► Familiarize yourself with the calculation of the phonon dispersion relations of diamond, following step-by-step the procedure outlined in Tutorial 4.1.

Exercise 2

► Repeat the calculations illustrated in Tutorial 4.1 for GaAs. Note that all these calculations refer to zone-center phonons, $\mathbf{q} \rightarrow 0$.

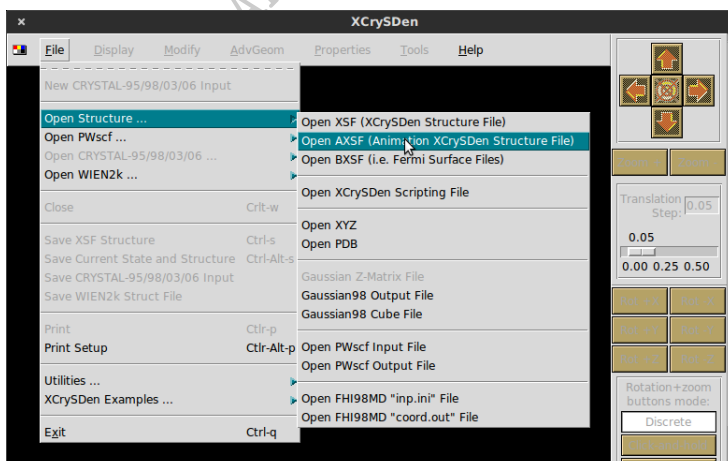
Exercise 3

If you succeeded to complete Exercise 2, 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.

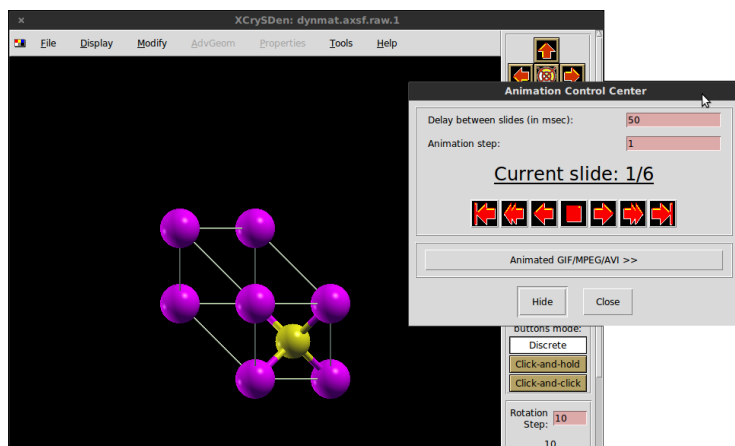
The file `dynmat.axsf` contains the vibrational eignemodes in a format which can be read and visualized by `xcrysden`. Let us recall that these modes correspond to the atomic displacement patterns associated with a wavevector $\mathbf{q} \rightarrow 0$ along the direction x (this was specified in the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

► In this exercise we want to visualize these modes.

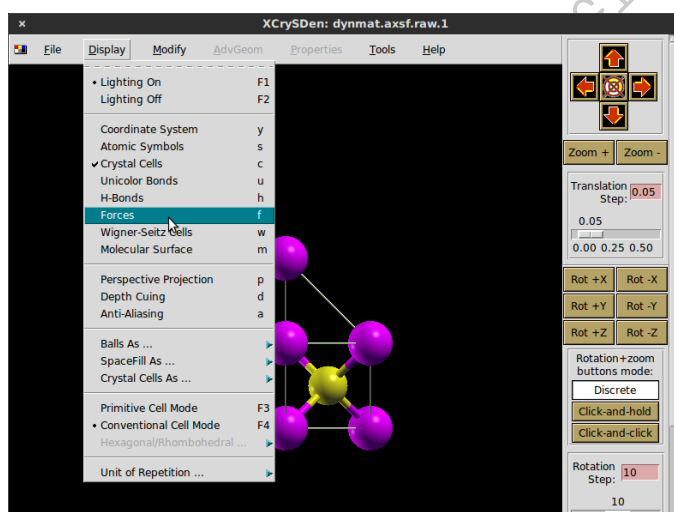
Let us call `xcrysden` and go through the following steps. We open the `dynmat.axsf` file:



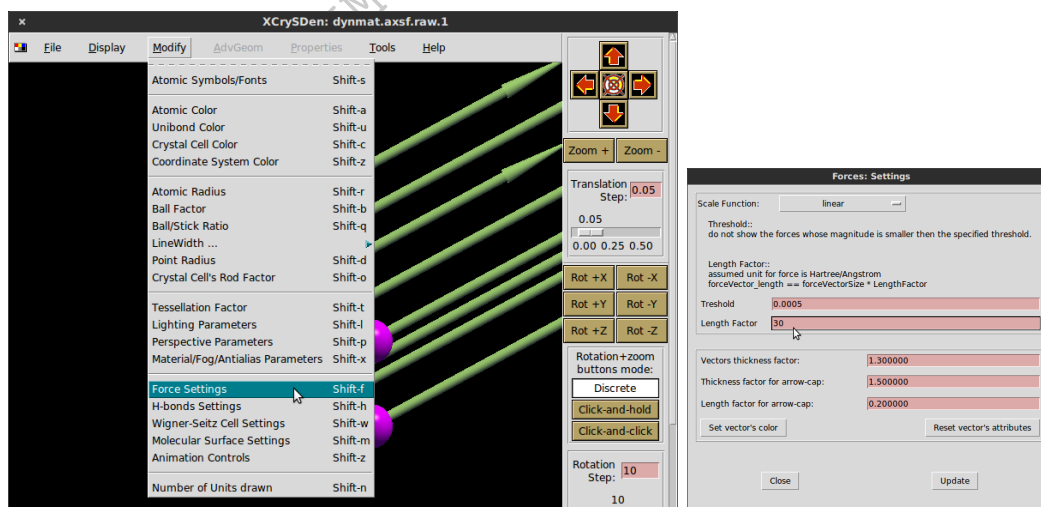
For the time being we ignore the window indicating the 'current slide'. This window will be used later to select the vibrational mode to be visualized.



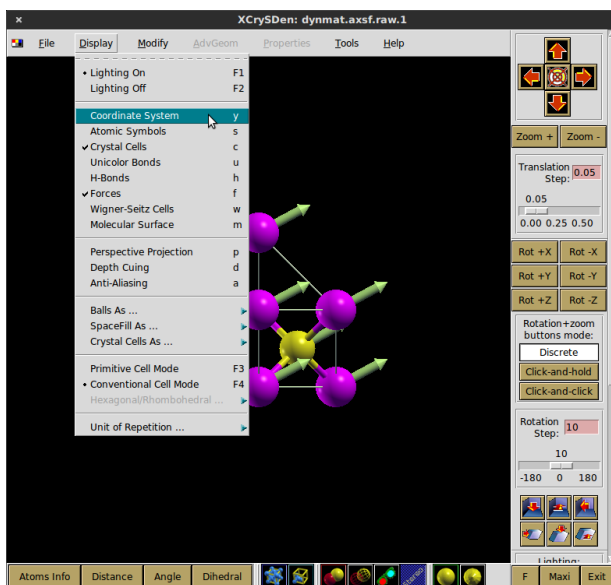
We activate the visualization of 'forces' (in our case the arrows will represent displacements, not forces, but the file format and the naming conventions are the same).



We adjust the length of the arrows by a uniform scaling

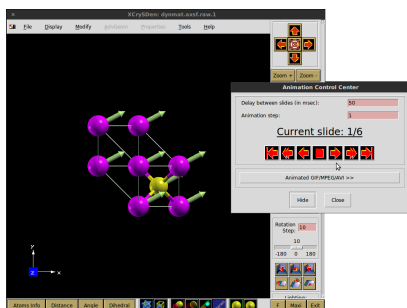


We ask the program to show the Cartesian axes.

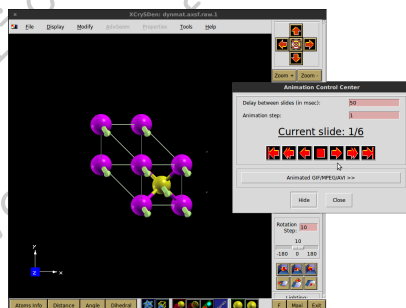


At this point we can use the window entitled 'Current Slide' to inspect each vibrational modes. The result should look similar to the following:

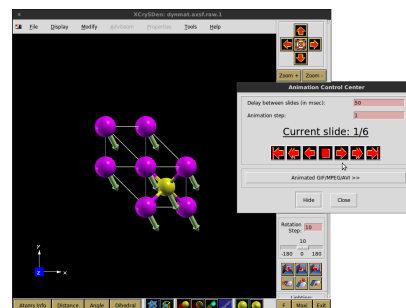
mode 1



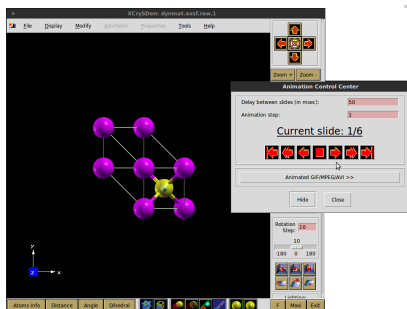
mode 2



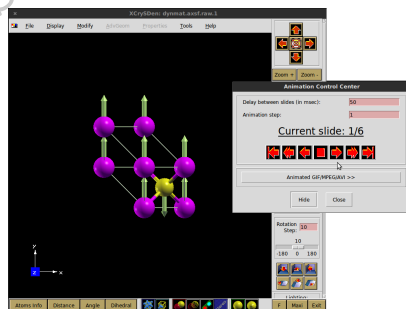
mode 3



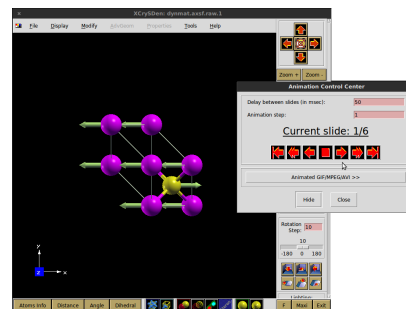
mode 4



mode 5



mode 6



Here we should be able to 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 `dynmat.in`), we can conclude that mode 6 is LO, while modes 4–5 are TO.

Exercise 4

- ▶ Calculate the phonon dispersion relations of GaAs, including the LO-TO splitting.

Note: 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 correct input file for `ph.x` is:

```
> cat > ph.in << EOF
phonons of GaAs
&inputph
  prefix = 'gaas',
  amass(1) = 69.723,
  amass(2) = 74.9216,
  epsilon = .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.

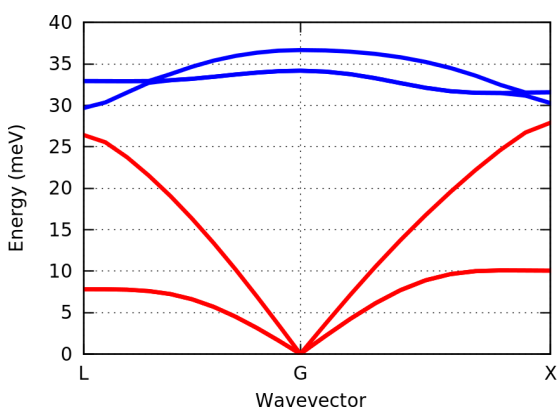
In order to obtain the dispersion relations, you will need to call `pw.x`, `ph.x`, `q2r.x`, and `matdyn.x` as we already did for diamond:

```
mpirun -np 12 pw.x -npool 4 <scf.in > scf.out
mpirun -np 12 ph.x -npool 4 <ph.in > ph.out
./q2r.x < q2r.in
./matdyn.x < matdyn.in
```

The input files `q2r.in` and `matdyn.in` must be prepared in the same way as for diamond. The input file `scf.in` for GaAs is the same as the one that we used in Tutorial 4.1.

- ▶ Plot the phonon dispersion relations of GaAs along the Brillouin zone path $L\Gamma X$ (as we did for diamond in Tutorial 4.1).
- ▶ Verify that the LO-TO splitting at Γ is the same as that calculated in Exercise 2.
- ▶ Compare your phonon dispersion relations with the inelastic neutron scattering data of [Strauch & Dörner, J. Phys. Condens. Matter 2, 1457 \(1990\)](#).

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



Exercise 5

In this exercise we want to calculate the phonon dispersion relations of cubic SrTiO₃, including the LO-TO splitting.

► We will use the input file `scf.in` from Tutorial 3.2 (with the optimized parameters given at the beginning of T3.2/Exercise 5), and the pseudopotentials from the same tutorial. Try to perform a test run using this setup, in order to make sure that everything goes smoothly.

► We now adapt to the case of SrTiO₃ the input file `ph.in` prepared in Exercise 4 for GaAs:

```
> cat > 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 SrTiO₃ 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. In these cases it is convenient to perform calculations in two steps:

- 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.
- 2) Once we are confident about the complete procedure, we launch a 'production' calculation with all the optimized convergence parameters. This may take up to 30min 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 SrTiO₃. The procedure is identical to what was done for GaAs on pag. 4.

Note: In this case we can plot the dispersions along the high-symmetry path $\Gamma XM\Gamma R$. 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 $2\pi/a$. The `matdyn.in` file will be:

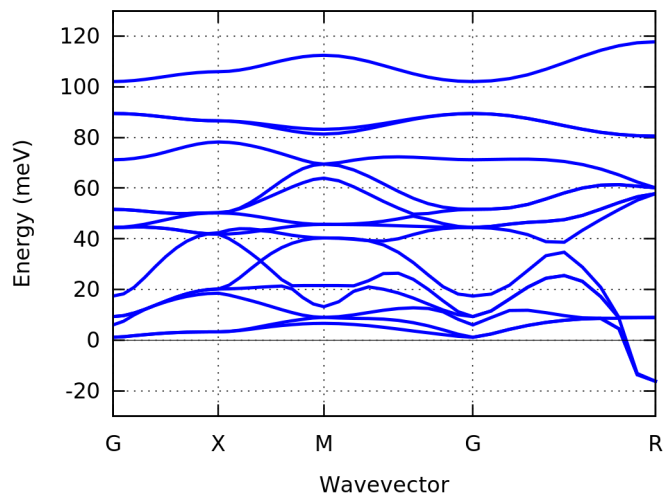
```
&input
asr = 'no',
flfrc = 'sto.fc',
flfrq = 'sto.freq'
/
41
```

```

0.000 0.000 0.000
0.050 0.000 0.000
0.100 0.000 0.000
0.150 0.000 0.000
0.200 0.000 0.000
0.250 0.000 0.000
0.300 0.000 0.000
0.350 0.000 0.000
0.400 0.000 0.000
0.450 0.000 0.000
0.500 0.000 0.000
0.500 0.050 0.000
0.500 0.100 0.000
0.500 0.150 0.000
0.500 0.200 0.000
0.500 0.250 0.000
0.500 0.300 0.000
0.500 0.350 0.000
0.500 0.400 0.000
0.500 0.450 0.000
0.500 0.500 0.000
0.450 0.450 0.000
0.400 0.400 0.000
0.350 0.350 0.000
0.300 0.300 0.000
0.250 0.250 0.000
0.200 0.200 0.000
0.150 0.150 0.000
0.100 0.100 0.000
0.050 0.050 0.000
0.000 0.000 0.000
0.050 0.050 0.050
0.100 0.100 0.100
0.150 0.150 0.150
0.200 0.200 0.200
0.250 0.250 0.250
0.300 0.300 0.300
0.350 0.350 0.350
0.400 0.400 0.400
0.450 0.450 0.450
0.500 0.500 0.500

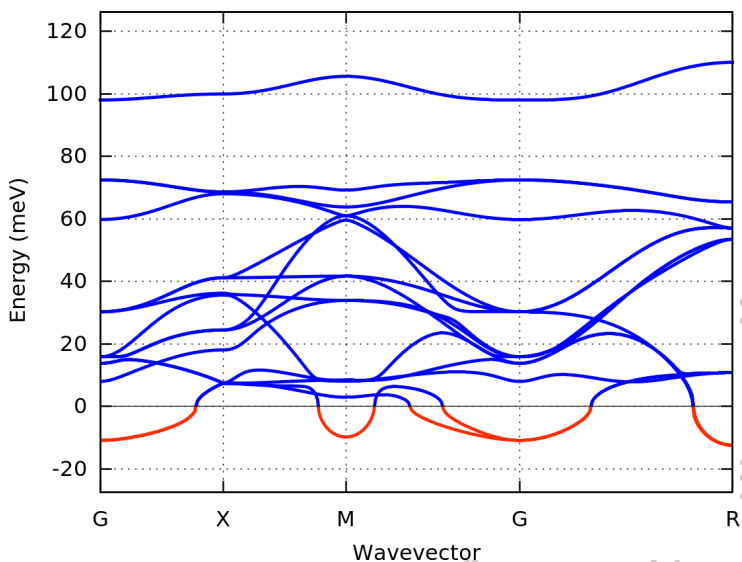
```

You should obtain something like the following:



► Now that we are confident about the entire procedure, we can perform a production run in order to 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`.

The final result should look as follows (in this plot a denser path of \mathbf{q} -points was used):



► 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 plot at the top of this page 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.

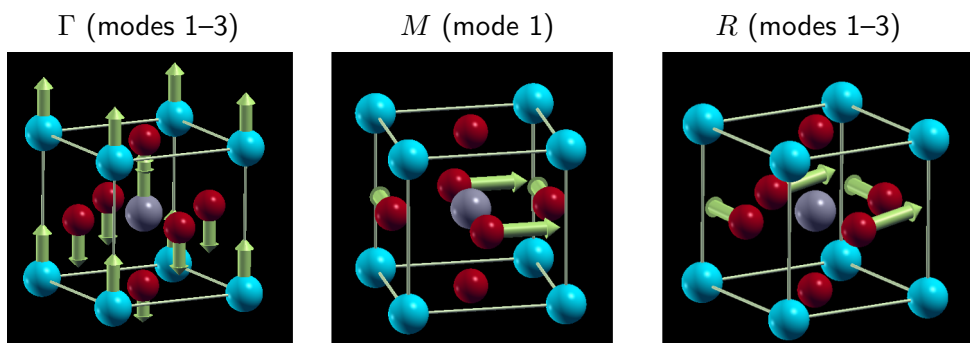
In order to better understand the origin of these instabilities, use `xcrysden` to visualize the displacement patterns of the soft modes at Γ , M , and R .

In order to use `xcrysden` you will need to generate `dynmat.axsf` files as in Exercise 3, using `dynmat.x`. The `dynmat.in` files should look as follows:

```
&input
  fildyn = 'dyn1'
/
```

where `dyn1` is for 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:



The soft modes at Γ indicate the presence of a ferroelectric (quantum paraelectric) instability, the soft modes at M and R indicate instabilities against rotations of the TiO_6 octahedra. The origin of these soft modes lies in that we are performing ground-state calculations (ie 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 `dynamt.x`.

In this case the appropriate input file is

```
&input
  fildyn = 'dyn1',
  asr = 'no',
  lperm = .true.,
  q(1)=1.0,
  q(2)=0.0,
  q(3)=0.0
/
```

Note that we are instructing `dynamt.x` to read data from `dyn1`, which corresponds to the Γ point. In this case we are also skipping the acoustic sum rule: the use of the sum rule is not meaningful when we have soft modes associated with structural instabilities.

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 should include temperature and quantum nuclear effects.

An introduction to density functional theory for experimentalists

Tutorial 5.1

```
$ cd ~/scratch/summerschool ; mkdir tutorial-5.1 ; cd tutorial-5.1
```

In this tutorial we will see how to calculate the band structures and the optical absorption spectra of semiconductors.

Band structures

We start from the band structure of **silicon**. First we copy the setup from Tutorial 2.1:

```
$ cp ../espresso-5.4.0/bin/pw.x ./
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/Si.pz-vbc.UPF
$ cat > scf.in << EOF
&control
  calculation = 'scf'
  prefix = 'silicon',
  pseudo_dir = './',
  outdir = './'
/
&system
  ibrav = 2,
  celldm(1) = 10.2094,
  nat = 2,
  ntyp = 1,
  ecutwfc = 25.0,
/
&electrons
  conv_thr = 1.0d-8
/
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. For this we submit a batch job with the line:

```
mpirun -np 12 pw.x -npool 1 < scf.in > scf.out
```

Now we want to calculate the band structure. This calculation is 'non self-consistent', in the sense that we use 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` determines the 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 > nscf.in << EOF
&control
  calculation = 'bands'
  prefix = 'silicon',
  pseudo_dir = './',
  outdir = './'
/
&system
 ibrav = 2,
  celldm(1) = 10.2094,
  nat = 2,
  ntyp = 1,
  ecutwfc = 25.0,
  nbnd = 8,
/
&electrons
  conv_thr = 1.0d-8
/
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 10
0.000 0.000 0.000 10
1.000 0.000 0.000 10
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 Tutorial 4.1. 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 (10) is the number of points in each segment.

So in this case we will have 10 points from $L = (1/2, 1/2, 1/2)2\pi/a$ to $\Gamma = (0, 0, 0)$ and 10 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` using our batch script:

```
mpirun -np 12 pw.x -npool 12 < nscf.in > nscf.out
```

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

```

...
  End of band structure calculation

    k = 0.5000 0.5000 0.5000 ( 568 PWs)  bands (ev):
-3.4467 -0.8437  4.9918  4.9918  7.7616  9.5416  9.5416 13.7947

    k = 0.4500 0.4500 0.4500 ( 571 PWs)  bands (ev):
-3.5870 -0.6409  5.0097  5.0097  7.7820  9.5628  9.5628 13.8281
...

```

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.

In order to plot the bands along the chosen path, we must extract manually these eigenvalues, and calculate the distance along the path as we move from L to Γ to X . We can do this quickly using the following `tcsh` script:

```
$ more extract.tcsh

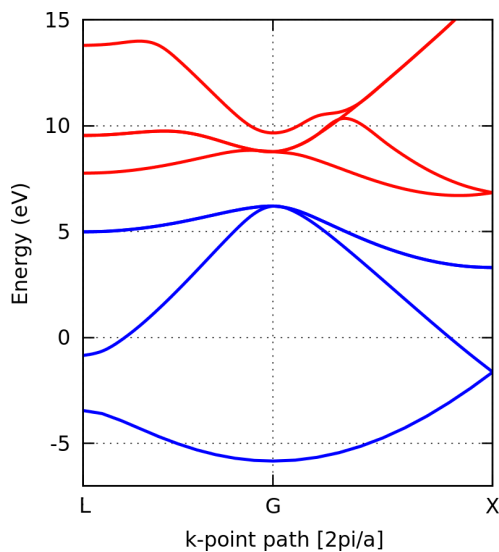
set klines = `grep -nr " k =" nscf.out | cut -d : -f 1`
set k0 = `head -$klines[1] nscf.out | tail -1`
set kk = 0
foreach NLINE ( $klines )
  set k = `head -$NLINE nscf.out | tail -1`
  @ NLINE = $NLINE + 2
  set eigenv = `head -$NLINE nscf.out | tail -1`
  set kk=`awk "BEGIN{print $kk +sqrt(($k[3]-$k0[3])^2+($k[4]-$k0[4])^2+($k[5]-$k0[5])^2)}"`
  set k0 = `echo $k`
  echo $kk $eigenv
end

$ tcsh extract.tcsh > bands.txt
```

At this point the file `bands.txt` will contain the distance along the **k**-point path and the eigenvalues in each column. We can plot this file using the following `gnuplot` instructions:

```
$ cat > plot.gp << EOF
unset key
set xlabel "k-point path [2pi/a]"
set xtics ("L" 0, "G" 0.866, "X" 1.866)
set ylabel "Energy (eV)"
set style line 1 lc 3 lw 2
plot "bands.txt" u 1:2 w l ls 1, \
    "bands.txt" u 1:3 w l ls 1, \
    "bands.txt" u 1:4 w l ls 1, \
    "bands.txt" u 1:5 w l ls 1, \
    "bands.txt" u 1:6 w l ls 1, \
    "bands.txt" u 1:7 w l ls 1, \
    "bands.txt" u 1:8 w l ls 1, \
    "bands.txt" u 1:9 w l ls 1
EOF
$ gnuplot
gnuplot> load "plot.gp"
```

The result should look similar to the following plot:



Here the conduction bands have been colored in red and a smoothing has been used via the option `smooth csplines` of `gnuplot`.

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 $E_g = 0.5128$ eV. The calculated band gap is much smaller than the experimental value of 1.2 eV.

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`. We compile this code as we already did for `pw.x` and `pp.x`:

```
$ cd ../espresso-5.4.0 ; make pp
$ cd ../tutorial-5.1 ; cp ../espresso-5.4.0/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:

```
&inputpp
  prefix = 'silicon'
  outdir = './',
  filplot = 'wavefc'
  plot_num = 7
  lsign = .true.
  kpoint = 11
  kband = 4
/
&plot
  iflag = 3
  output_format = 5
  fileout = 'silicon.xsf'
/
```

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 k_{point} and k_{band} indicate the k -point and band that we want to plot. In this case we are choosing the 11-th point from the list on page 2 and the band number 4. This is precisely the valence band top at Γ . The flags $\text{iflag} = 3$ and $\text{output_format} = 5$ specify that we want a 3D plot and that this must be in `xcrysden` format, respectively.

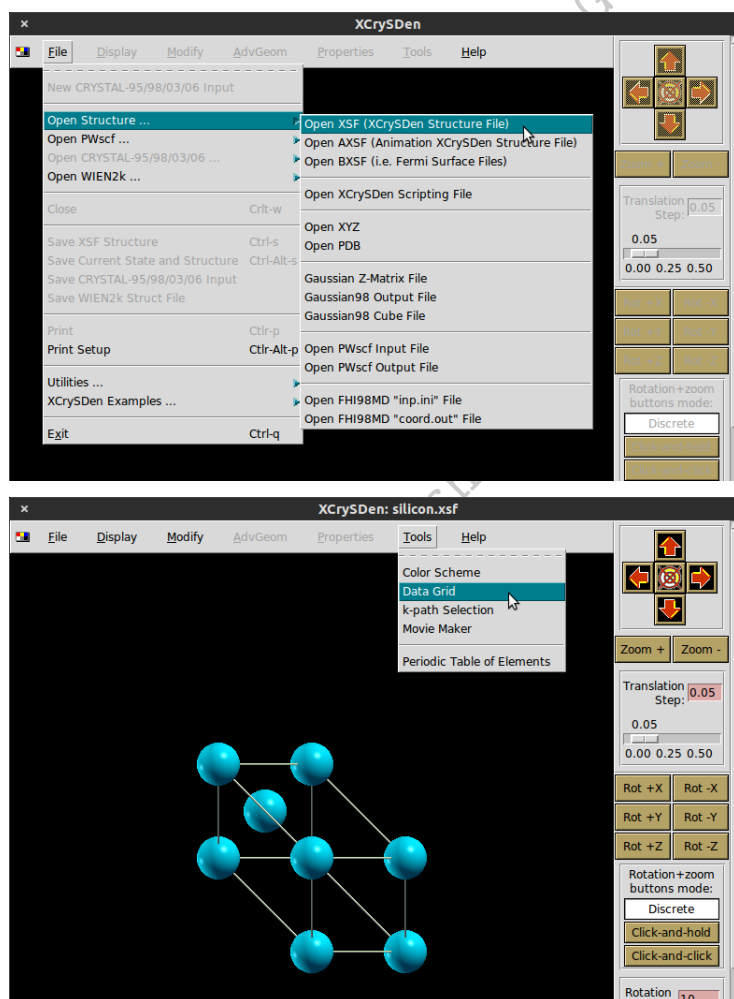
There are many other options for plotting other quantities of interest, for the complete range please see the documentation page:

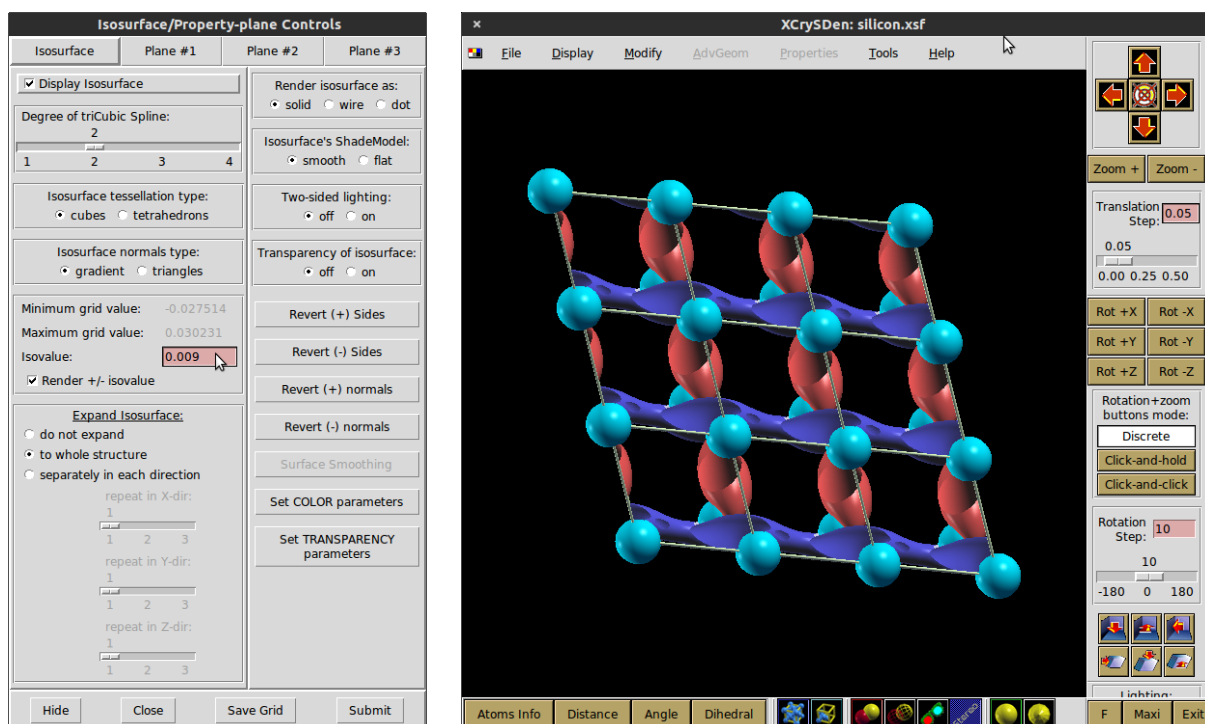
http://www.quantum-espresso.org/wp-content/uploads/Doc/INPUT_PP.html

In order to obtain our wavefunction, first we execute `pw.x` from the previous section, and then we run `pp.x` using the same number of CPUs and in the same batch script:

```
mpirun -np 12 pw.x < nscf.in
mpirun -np 12 pp.x < 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.

Calculation of UV/Vis spectra

Now we consider the band structure and the optical absorption spectrum of **GaAs**. We already studied GaAs in Tutorial 4.1, therefore we can start from the same `scf.in` file:

```
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/Ga.pz-bhs.UPF
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/As.pz-bhs.UPF
$ cat > scf.in << EOF
&control
  calculation = 'scf'
  prefix = 'gaas',
  pseudo_dir = './',
  outdir = './'
/
&system
 ibrav = 2,
  cellldm(1) = 10.4749,
  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:

```
mpirun -np 12 pw.x -npool 12 < scf.in > scf.out
```

Now we can take a look at the band structure of GaAs. The procedure is identical to what we just did for silicon, and we can recycle most of the input file `nscf.in` from pag. 1. The colored lines below indicate the modifications required to work with GaAs instead of Si. These parameters are taken directly from the input file `scf.in` above:

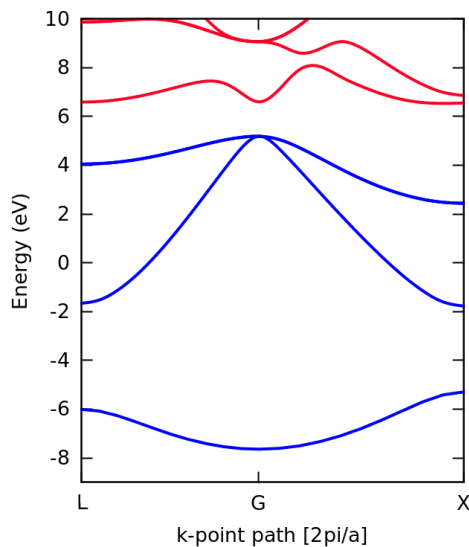
```
$ cat > nscf.in << EOF
&control
  calculation = 'bands'
  prefix = 'gaas',
  pseudo_dir = './',
  outdir = './'
/
&system
  ibrav = 2,
  celldm(1) = 10.4749,
  nat = 2,
  ntyp = 2,
  ecutwfc = 40.0,
  nbnd = 8,
/
&electrons
  conv_thr = 1.0d-8
/
ATOMIC_SPECIES
  Ga 1.0 Ga.pz-bhs.UPF
  As 1.0 As.pz-bhs.UPF
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 10
0.000 0.000 0.000 10
1.000 0.000 0.000 10
EOF
```

We can now run the band structure calculation, precisely as we did for silicon:

```
mpirun -np 12 pw.x < scf.in > scf.out
mpirun -np 12 pw.x < nscf.in > nscf.out
```

At the end of the run we extract the **k**-point path and the Kohn-Sham eigenvalues again using the script `extract.tcsh`, and plot these data using `plot.gp`.

The resulting band structure should look as follows (this plot has also been smoothed and the color of the conduction bands has been modified):



In this calculation we see that the direct gap at Γ is $E_g = 1.42$ eV. This value is unusually close to experiment (1.52 eV) for a DFT/LDA calculation. We can also see that in this calculation we have an indirect gap of 1.35 eV along the ΓX line. This is an artifact of the DFT/LDA approximation (GaAs is a direct-gap semiconductor).

Now we calculate the imaginary part of the dielectric function, $\epsilon_2(\omega)$, which is related to the optical absorption coefficient $\kappa(\omega)$ by:

$$\kappa(\omega) = \frac{\omega \epsilon_2(\omega)}{c n(\omega)}$$

where $\hbar\omega$ is the photon energy, c the speed of light, and n the refractive index.

In order to calculate $\epsilon_2(\omega)$ we use the post-processing code `epsilon.x`. We already compiled this program when we issued `make pp` on pag. 4, therefore we only need to copy the code inside the current directory:

```
$ cp ../espresso-5.4.0/bin/epsilon.x ./
```

The manual of this post-processing code can be found in the directory `../espresso-5.4.0/PP/Doc`. To obtain a PDF version we simply issue:

```
$ cd ../espresso-5.4.0/PP ; make doc
```

The as-compiled PDF file `eps_man.pdf` will be found in the directory `espresso-5.4.0/PP/Doc`.

The input file for `epsilon.x` is as follows:

```
$ cat > eps.in << EOF
&inputpp
  outdir = './'
  prefix = 'gaas'
  calculation = 'eps'
/
&energy_grid
  smear_type = '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 that we have seen in Lecture 5.1. 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 > scf_eps.in << EOF
&control
  calculation = 'scf'
  prefix = 'gaas',
  pseudo_dir = './',
  outdir = './'
/
&system
  ibrav = 2,
  cellldm(1) = 10.4749,
  nat = 2,
  ntyp = 2,
  ecutwfc = 40.0,
  nosym = .true.
  nbnd = 20
/
&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 crystal
  64
  0.120.120.120.016
  0.120.120.380.016
  0.120.120.620.016
  0.120.120.880.016
  0.120.380.120.016
  0.120.380.380.016
  0.120.380.620.016
  0.120.380.880.016
  0.120.620.120.016
  0.120.620.380.016
  0.120.620.620.016
  0.120.620.880.016
  0.120.880.120.016
  0.120.880.380.016
  0.120.880.620.016
  0.120.880.880.016
  0.380.120.120.016
  0.380.120.380.016
  0.380.120.620.016
  0.380.120.880.016
  0.380.380.120.016
  0.380.380.380.016
  0.380.380.620.016
  0.380.380.880.016
  0.380.620.120.016
  0.380.620.380.016
  0.380.620.620.016
  0.380.620.880.016
```

```
0.380.880.120.016
0.380.880.380.016
0.380.880.620.016
0.380.880.880.016
0.620.120.120.016
0.620.120.380.016
0.620.120.620.016
0.620.120.880.016
0.620.380.120.016
0.620.380.380.016
0.620.380.620.016
0.620.380.880.016
0.620.620.120.016
0.620.620.380.016
0.620.620.620.016
0.620.620.880.016
0.620.880.120.016
0.620.880.380.016
0.620.880.620.016
0.620.880.880.016
0.880.120.120.016
0.880.120.380.016
0.880.120.620.016
0.880.120.880.016
0.880.380.120.016
0.880.380.380.016
0.880.380.620.016
0.880.380.880.016
0.880.620.120.016
0.880.620.380.016
0.880.620.620.016
0.880.620.880.016
0.880.880.120.016
0.880.880.380.016
0.880.880.620.016
0.880.880.880.016
EOF
```

The modifications brought to our standard input file are as follows:

- We explicitly provide a uniform grid of **k**-points
- We turn off the automatic reduction of **k**-points that `pw.x` does by using crystal symmetries
- We request a number of bands much larger than the number of valence bands, 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.

The grid used in the above input file is a uniform and shifted $4 \times 4 \times 4$ mesh in the Brillouin zone (64 points).

We can now execute `pw.x` and `epsilon.x` using the following lines in our batch script:

```
mpirun -np 12 pw.x -npool 12 < scf_eps.in > scf_eps.out
mpirun -np 12 epsilon.x -npool 12 < eps.in > eps.out
```

At the end of the execution we will find the output files:

```
$ more epsr.dat
```

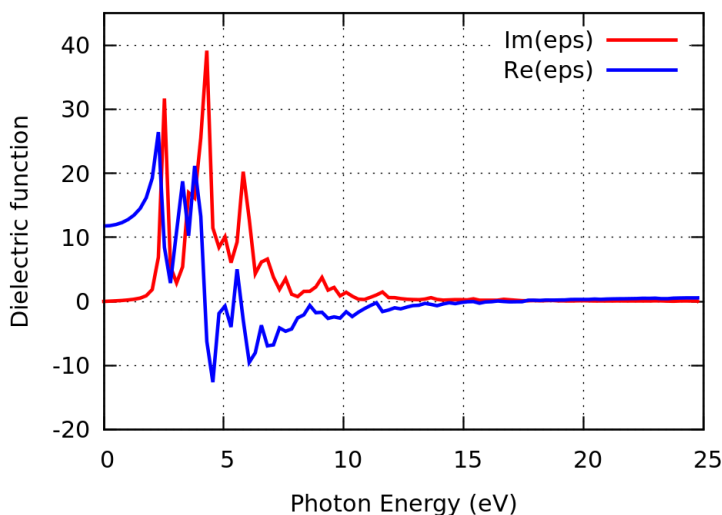
```
# energy grid [eV]      epsr_x  epsr_y  epsr_z
  0.000000          11.787553  11.787537  11.787546
  0.060120          11.790677  11.790662  11.790670
...
```

```
$ more epsi.dat
```

```
# energy grid [eV]      epsi_x  epsi_y  epsi_z
  0.000000           0.000000   0.000000   0.000000
  0.060120           0.010446   0.010446   0.010446
...
```

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. In this case the system is cubic, therefore the x , y and z components will be identical.

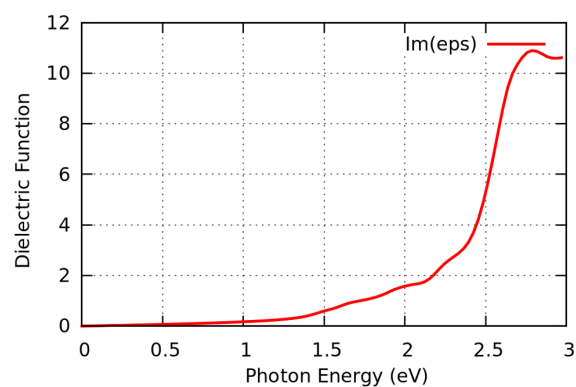
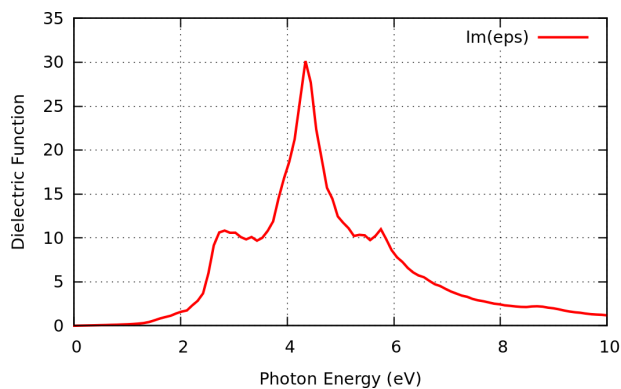
A plot of these quantities using `gnuplot` yields:



From this figure we can read the high-frequency dielectric constant of GaAs, $\epsilon_\infty = 11.8$. This value is similar but not identical to what we obtained in Tutorial 4.1 (11.6). This difference is due to the different sampling of the Brillouin zone and the different computational technique employed here.

In the above plot 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 $4 \times 4 \times 4$ mesh, and this is definitely not enough for studying the dielectric function. The meshes required for calculations of dielectric functions may need to contain as many as $50 \times 50 \times 50$ points. With our coarse $4 \times 4 \times 4$ mesh it is difficult to identify the optical absorption onset around the direct band gap at 1.4 eV.

A more accurate calculation using a $30 \times 30 \times 30$ mesh is shown below, together with a zoom where we can see the onset around 1.4 eV (broadened by the Gaussian smearing, which is set to `intersmear = 0.2 eV`):



Note. The calculation of dielectric functions by means of `epsilon.x` suffers from two important approximations, namely the 'independent-particle approximation' and the neglect of the 'nonlocal component' of the pseudopotentials. As a result, while the gross structure of the spectrum is reasonably accurate, subtle features such as the intensity and energy of the peaks are not very reliable.

A more comprehensive discussion of dielectric functions in DFT can be found in Chapter 11 of the Book.

Prof. Felicia
University of Oxford
PARADIM School · Cornell, Ju.

An introduction to density functional theory for experimentalists

Tutorial 5.2

Hands-on session

```
$ cd ~/scratch/summerschool; mkdir tutorial-5.2 ; cd tutorial-5.2
```

In this tutorial we will calculate the band structures of **GaAs**, **SrTiO₃**, and **graphite**. Then we will test a calculation of dielectric functions for the case of GaAs.

In the last exercise we examine the performance of the LDE and PBE exchange and correlation in predicting the interlayer binding energy and the interlayer distance in graphite.

Exercise 1

► Familiarize yourself with the calculation of band structures, following step-by-step the example of GaAs discussed in Tutorial 5.1.

Exercise 2

In this exercise we want to calculate the band structure of SrTiO₃.

► Perform a test run for SrTiO₃ using exactly the same setup as in Exercise 5 of Tutorial 3.2.

As a sanity check, you should obtain a total energy of -105.4136 Ry.

► Now we run a band structure calculation for SrTiO₃. Prepare the input file `nscf.in` for the non-selfconsistent calculation, using the high-symmetry path $\Gamma XM\Gamma R$. 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 $2\pi/a$.

As a reference, the input file should look like this:

```
&control
  calculation = 'bands'
  prefix = 'sto',
  pseudo_dir = './',
  outdir = './'
/
&system
  ibrav = 1,
  celldm(1) = 7.18899,
  nat = 5,
  ntyp = 3,
  ecutwfc = 210.0,
  nbnd = 20,
/
&electrons
/
ATOMIC_SPECIES
Sr 1.0 Sr.pz-hgh.UPF
Ti 1.0 Ti.pz-hgh.UPF
```

```

0 1.0 0.pz-hgh.UPF
ATOMIC_POSITIONS crystal
Sr 0.0 0.0 0.0
Ti 0.5 0.5 0.5
O 0.5 0.0 0.5
O 0.5 0.5 0.0
O 0.0 0.5 0.5
K_POINTS tpiba_b
5
0.0 0.0 0.0 10
0.5 0.0 0.0 10
0.5 0.5 0.0 10
0.0 0.0 0.0 10
0.5 0.5 0.5 10

```

Note that the number of bands must be increased in order to calculate conduction states (here we have set `nbnd = 20`).

► After executing `pw.x` with this input file, you can extract the Kohn-Sham eigenvalues and the coordinate along the **k**-path using the following `tcsh` script:

```

$ more extract2.tcsh

set klines = `grep -nr " k =" nscf.out | cut -d : -f 1`
set k0 = `head -$klines[1] nscf.out | tail -1`
set kk = 0
foreach NLINE ( $klines )
  set k = `head -$NLINE nscf.out | tail -1`
  @ NLINE = $NLINE + 4
  set eigenv = `head -$NLINE nscf.out | tail -3`
  set kk=`awk "BEGIN{print $kk +sqrt(($k[3]-$k0[3])^2+($k[4]-$k0[4])^2+($k[5]-$k0[5])^2)}"`
  set k0 = `echo $k`
  echo $kk $eigenv
end

$ tcsh extract2.tcsh > bands.txt

```

The band structure in `bands.txt` can be plotted for example using `gnuplot`:

```

$ cat > plot_ST0.gp << EOF
unset key
set xlabel "Wavevector [2pi/a]"
set xtics ("G" 0.0, "X" 0.500, "M" 1.000, "G" 1.707, "R" 2.573)
set ylabel "Energy (eV)"
set style line 1 lc 3 lw 2
set grid
plot [] [0:11] \\  

    "bands.txt" u 1:2 w l ls 1, \  

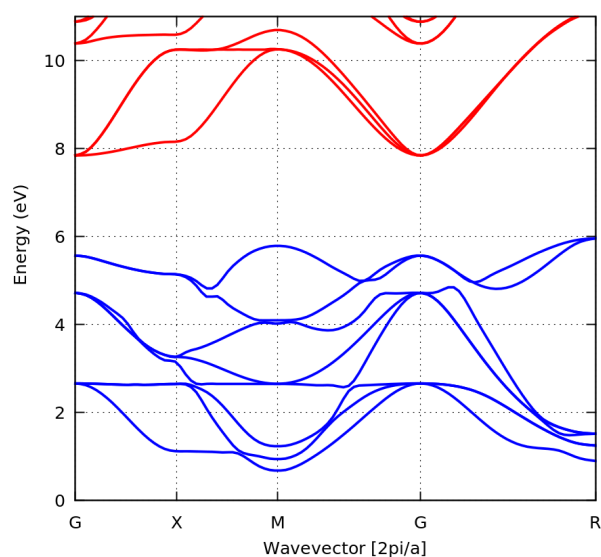
    "bands.txt" u 1:3 w l ls 1, \  

    "bands.txt" u 1:4 w l ls 1, \  

    ...
EOF
$ gnuplot
gnuplot> load "plot_ST0.gp"

```

As a reference, the plot should look similar to the following:



► Using the band structure just calculated, determine the lowest direct and indirect band gaps of SrTiO_3 in DFT/LDA.

► Compare your band gaps and band structure with previous experimental and theoretical work from [Bentham et al, J. Appl. Phys. 90, 6156 \(2001\)](#) and from [Benrekia et al, Physica B 407, 2632 \(2012\)](#).

Exercise 3

In this exercise we calculate the band structure of graphite.

► Perform a test run for the total energy of graphite in the ground state, using the optimized setup determined in Exercise 3 of Tutorial 2.2.

As a reminder, the optimized input file `scf.in` is:

```
&control
  calculation = 'scf'
  prefix = 'graphite',
  pseudo_dir = './',
  outdir = './'
/
&system
 ibrav = 4,
  celldm(1) = 4.60913,
  celldm(3) = 2.729,
  nat = 4,
  ntyp = 1,
  ecutwfc = 100,
/
&electrons
  conv_thr = 1.0d-8
/
ATOMIC_SPECIES
C 1.0 C.pz-vbc.UPF
ATOMIC_POSITIONS crystal
C 0.00 0.00 0.25
C 0.00 0.00 0.75
```

```

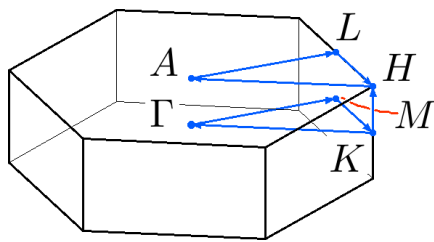
C 0.333333 0.666666 0.25
C 0.666666 0.333333 0.75
K_POINTS automatic
6 6 2 1 1 1

```

► Prepare the input file `nscf.in` for the band structure calculation along the high-symmetry path $K \rightarrow \Gamma \rightarrow M \rightarrow K \rightarrow H \rightarrow A \rightarrow L \rightarrow H$.

In this case we want to perform calculations for 16 bands.

The location of the high-symmetry points in the Brillouin zone of graphite is given below:



In order to generate the **k**-point path we can use `xcrysden`. After loading the input file `scf.in` inside `xcrysden`, we go through the following steps:

#	reciprocal coordinates	label
1	0.66667 -0.33333 0.00000	
2	0.00000 0.00000 0.00000	
3	0.50000 0.00000 0.00000	
4	0.66667 -0.33333 0.00000	
5	0.66667 -0.33333 0.50000	
6	0.50000 0.00000 0.50000	
7	0.00000 0.00000 0.50000	
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		

At the end of the file `myfile.kpf` we will find the desired path. Here each vertex is expressed in **crystal coordinates**, that is in terms of the primitive vectors of the reciprocal lattice.

```
$ more myfile.kpf
```

...

```
#
# #-----#
# # REAL FORM of k-point COORDINATES #
# #-----#
#
```

```
Real form of k-point coordinates (kx,ky,kz,label):
 0.6666666667 -0.3333333333 0.0000000000 K.1
 0.0000000000 0.0000000000 0.0000000000 K.2
 0.5000000000 0.0000000000 0.0000000000 K.3
 0.6666666667 -0.3333333333 0.0000000000 K.4
 0.6666666667 -0.3333333333 0.5000000000 K.5
 0.0000000000 0.0000000000 0.5000000000 K.6
 0.5000000000 0.0000000000 0.5000000000 K.7
 0.6666666667 -0.3333333333 0.5000000000 K.8
```

...

Since the coordinates of the vertices are expressed in terms of the reciprocal lattice vectors, in our input file we will need to replace `K_POINTS tpiba_b` by `K_POINTS crystal_b`. Please see the related [online documentation](#).

As a reference, the correct input file should contain the following lines:

```
$ more nscf.in
```

...

```
K_POINTS crystal_b
8
 0.6666666667 -0.3333333333 0.0000000000 10
 0.0000000000 0.0000000000 0.0000000000 10
 0.5000000000 0.0000000000 0.0000000000 10
 0.6666666667 -0.3333333333 0.0000000000 10
 0.6666666667 -0.3333333333 0.5000000000 10
 0.0000000000 0.0000000000 0.5000000000 10
 0.5000000000 0.0000000000 0.5000000000 10
 0.6666666667 -0.3333333333 0.5000000000 10
```

Once executed `pw.x` using the input file `nscf.in`, we can extract the bands using exactly the same script `extract2.tcsh` as in Exercise 2.

► Plot the band structure of graphite using `gnuplot`.

A possible `gnuplot` script for this is:

```
unset key
set xlabel "Wavevector [2pi/a]"
set xtics ("K" 0.0, "G" 0.6667, "M" 1.24406, "K" 1.57744, "H" 1.76064, "A" 2.42734, "L" 3.00469, "H" 3.33807)
set ylabel "Energy (eV)"
set style line 1 lc 3 lw 2
set grid
plot [0:3.33807] [:20] \
 "bands.txt" u 1:2 w l ls 1, \
 "bands.txt" u 1:3 w l ls 1, \
 ...
 "bands.txt" u 1:16 w l ls 1, \
 "bands.txt" u 1:17 w l ls 1
```

-
- Compare your calculated band structure with the results of [Marinopoulos et al, Phys. Rev. B 69, 245419 \(2004\)](#).

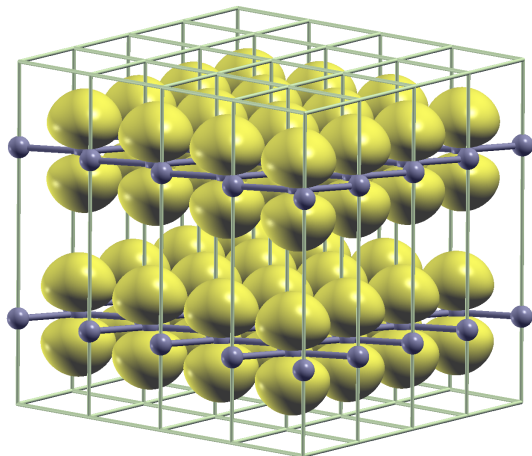
Exercise 4

- Using the post-processing program `pp.x`, verify that the wavefunctions near the Dirac point are p_z orbitals as expected (the Dirac point is the K point, at the intersection between valence and conduction bands).

We can run `pp.x` using the following input file, after having specified the appropriate \mathbf{k} -point and band:

```
&inputpp
  prefix = 'graphite'
  outdir = './',
  filplot = 'wavefc'
  plot_num = 7
  kpoint = ...
  kband = ...
/
&plot
  iflag = 3
  output_format = 5
  fileout = 'graphite.xsf'
/
```

By tuning the isosurface value you should be able to obtain something similar to the following:



Exercise 5

In this exercise we want to calculate the optical absorption spectrum of GaAs, precisely as we did in Tutorial 5.1.

- Repeat the steps on pagg. 8–11 of Tutorial 5.1 in order to calculate the absorption spectrum of GaAs.
- Run `epsilon.x` once again, this time using a smearing parameter `intersmear = 0.02 eV`.

► Compare the imaginary part of the dielectric function of GaAs, $\epsilon_2(\omega)$, obtained above using the two smearing values 0.1 eV and 0.2 eV.

Note. In `gnuplot` we can do this by using the usual `plot` command:

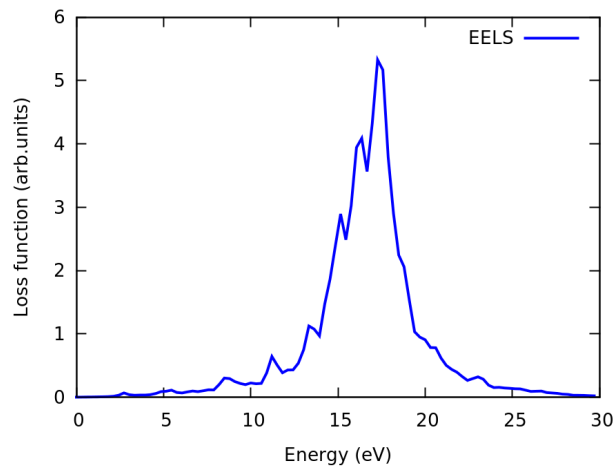
```
gnuplot> plot "epsi_0.02.dat" u 1:2 w l, "epsi_0.2.dat" u 1:2 w l
```

where `epsi_0.02.dat` and `epsi_0.2.dat` indicate the output files of the two separate runs.

Here you should see that, as we reduce the artificial smearing, the curve turns into a series of discrete peaks. This is because we are discretizing the Brillouin zone using only 64 points. An accurate calculation requires a much finer sampling.

► Plot the energy loss function of GaAs, calculated using a Gaussian smearing of 0.5 eV, for energies between 0 and 30 eV. The loss function is given by $-\text{Im} \epsilon^{-1}(\omega)$, and can be found in the output file `eels.dat`.

You should obtain something looking like the following:



The energy of the peak in this plot corresponds to the valence (bulk) plasmon of GaAs.

Exercise 6

In this exercise we want to explore the sensitivity of the structure of **graphite** to the exchange and correlation functional used in the calculation.

The structure of graphite and the various calculation parameters have already been optimized in Tutorial 2.2, and the optimized input file was given earlier, see pag. 3.

► Calculate the total energy of graphite within DFT/LDA as a function of interlayer distance, for ratios c/a ranging between 2.2 and 3.2.

For this exercise you can simply use the `scf.in` file on pag. 3 and perform several calculations for different values of `celldm(3)`.

► Now convert the results of the previous step into a cohesive energy, and plot the cohesive energy as a function of the c/a ratio.

Note: The total energy of an isolated C atom using the same parameters as for graphite is -10.73321495 Ry.

► Now we repeat the same calculations as in the previous steps, this time using the PBE exchange and correlation functional. Determine the cohesive energy of graphite within DFT/PBE, as a function of the ratio c/a , for the same range considered above.

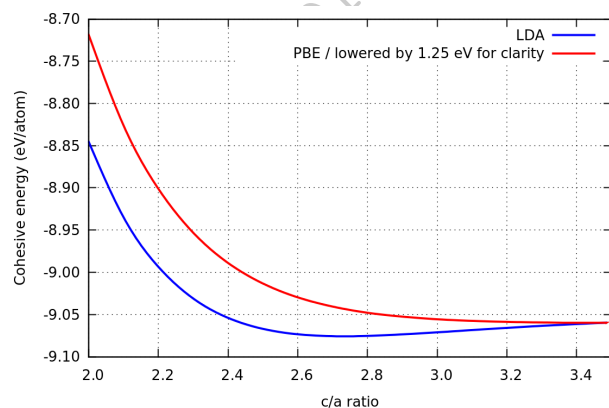
Note: For this exercise we need a new pseudopotential, constructed within the PBE approximation. The following pseudopotential has been tested separately:

```
$ wget http://www.quantum-espresso.org/wp-content/uploads/upf_files/C.pbe-hgh.UPF
```

It was found that the kinetic energy cutoff required to have total energies converged to within 10 meV is `ecutwfc = 130` Ry. Furthermore, the total energy of an isolated C atom using this pseudopotential was calculated to be -10.82157608 Ry.

► Plot the cohesive energies just obtained, and compare the predictions of LDA and PBE. What should we conclude from this comparison?

For your reference, the plot should look as follows:



Graphite is a prototypical system where van der Waals interactions play an important role in the interlayer binding.