Lab 2: Handout
Quantum-Espresso: a first-principles code

We will be using the Quantum-Espresso package as our first-principles code. Quantum-Espresso is a full *ab initio* package implementing electronic structure and energy calculations, linear response methods (to calculate phonon dispersion curves, dielectric constants, and Born effective charges) and third-order anharmonic perturbation theory. Quantym-Espresso also contains two molecular-dynamics codes, CPMD (Car-Parrinello Molecular Dynamics) and FPMD (First-Principles Molecular Dynamics). Inside this package, PWSCF is the code we will use to perform total energy calculations. PWSCF uses both norm-conserving pseudopotentials (PP) and ultrasoft pseudopotentials (US-PP), within density functional theory (DFT).

Quantum-Espresso is free under the conditions of the GNU GPL. Further information (including online manual) can be found at the Quantum-Espresso website http://www.quantum-espresso.org or http://www.pwscf.org/. Quantum-Espresso is currently at version 2.1.2. and is under very active development.

There are many other first-principles codes that one can use. These have been listed in the lecture slides, and include:

ABINIT
http://www.abinit.org
DFT Plane wave pseudo-potential (PP) code. ABINIT is also distributed under the GPL license.

CPMD
http://www.cpmd.org
DFT Plane wave pseudopotential code. Car-Parrinello molecular dynamics. GPL.

SIESTA
http://www.uam.es/departamentos/ciencias/fismateriac/siesta/
DFT in a localized atomic base.

VASP
http://cms.mpi.univie.ac.at/vasp/
DFT Ultrasoft pseudo-potential (US-PP) and PAW code. US PP's are faster and more complex than corresponding PP codes, with similar accuracy. Moderate cost for academics (~$2000)

WIEN2k
http://www.wien2k.at/
DFT Full-Potential Linearized Augmented Plane-Wave (FLAPW). FLAPW is the most accurate implementation of DFT, but the slowest. Small cost for academics (~$500)

Gaussian
http://www.gaussian.com
Quantum Chemistry Code - includes Hartree-Fock and higher-order correlated electrons approaches. Moderate cost for academics (~$3000). Has recently been extended to solids.

Crystal
http://www.cse.clrc.ac.uk/cmg/CRYSTAL
HF and DFT code. Small cost for academics (~$500).

This is a tutorial on how to get **energies** using the PWSCF code in Quantum-Espresso.

Some helpful conversions:

1 **bohr** = 1 **a.u.** (atomic unit) = 0.529177249 **angstroms**.
1 **Rydberg** (R_∞) = 13.6056981 **eV**
1 **eV** =1.60217733 x 10^{-19} **Joules**

From athena, log into hpcbeo2

```
$ ssh hpcbeo2.mit.edu
```

Next create a directory to store your files (ask if you need help). For example (assuming you've not already created a directory called ("3.320"),

```
$ mkdir 3.320
$ cd 3.320
$ mkdir LAB2
$ cd LAB2
$ mkdir PROBLEM1
```

Now copy over the appropriate input file and script that we will use

```
$ cp ~mounet/LAB2/C.scf.in .
$ cp ~mounet/LAB2/C.scf.j .
```

You can also copy the executable (not mandatory, since the script is looking for it directly into the directory ~mounet/LAB2/)

```
$ cp ~mounet/LAB2/pw.x .
```

Problem 1 concerns convergence issues in first-principles calculations. Below is some background for problem 1A and 1B; if you do not think you need it, skip to the "summary of background" section.

Background for problem 1A

Remember that we are dealing with infinite systems using periodic boundary conditions. This means that we can use the Bloch theorem to help us solve the Schrödinger equation. The Bloch theorem says:

$$\psi_{n\vec{k}}(\vec{r}) = \exp(i\vec{k}\cdot\vec{r})\, u_{n\vec{k}}(\vec{r})$$

with

$$u_{n\vec{k}}(\vec{r}) = \sum_{G} c_G \exp(i\vec{G}\cdot\vec{r})$$

In these equations, $\psi_{n\vec{k}}(\vec{r})$ is the wavefunction, $u_{n\vec{k}}(\vec{r})$ is a function that is periodic in the same way as the lattice, $\vec{G}$ sums over all (at least in principle) reciprocal lattice vectors, and c_G's are coefficients in the expansion. In this case, our *basis functions* (ie what we expand in) are plane waves. They are called "plane waves" because surfaces of constant phase are parallel *planes* perpendicular to the direction of propagation. Remember that limiting the plane wave expansion to the infinite, but numerable and discrete set of $\vec{G}$ vectors that are integer multiples of the three primitive lattice vectors, we are selecting plane waves that have a periodicity compatible with the periodic boundary conditions of our direct lattice.

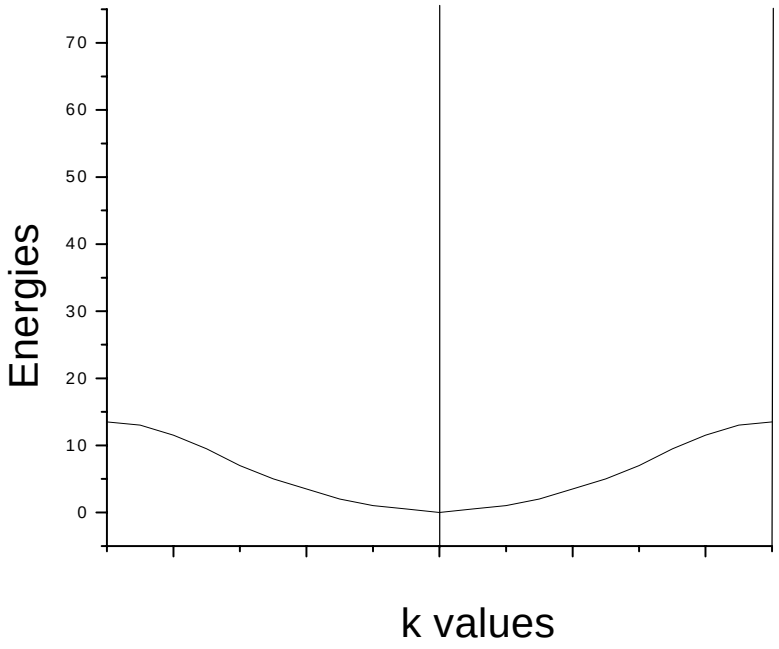
In actual calculations, we have to limit the plane wave expansion at some point (i.e. stop taking more $\vec{G}$'s). This is called the *planewave cutoff*. Cutoffs are always given in energy units (such as Rydberg or eV) corresponding to the kinetic energy of the highest $\vec{G}$.

Note: The units of reciprocal lattice are the inverse of the direct lattice, or 1/length. However, we can convert 1/length to energy units (Remember $\lambda\nu = c$, and $h\nu = E$. λ is wavelength, and is in units of 1/length)

Problem 1a is designed to test cutoff convergence issues. You can always take a higher cutoff than you need, but the calculations will take longer.

Background for Problem 1B.

Because of the Bloch theorem, we need to solve a Schrödinger-like Kohn-Sham equation (i.e. iterate selfconsistently the diagonalization of a MxM matrix, where M is the number of basis functions) everywhere in the Brillouin Zone. In practice, we do it for a finite number of $\vec{k}$ values (e.g. the coarse grained Monkhorst-Pack mesh), and get a value for E at each $\vec{k}$. This is seen in the schematic below.



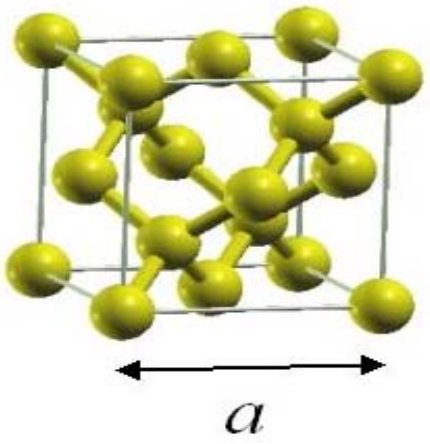
The picture goes over the first Brillouin zone. (If you don't know what a Brillouin zone is, it's time to go over the concepts of direct and reciprocal lattice). To obtain a value for *E*, the energy of the crystal, we need to integrate over the first Brillouin zone, where the bands are occupied (and divide by the volume). Thus, summing over a finite number of $\vec{k}$ points is an approximation to performing an integral. You will need to make sure you have enough $\vec{k}$-points to have a converged value for the energy.

Summary of background

For all first-principles calculations, you must pay attention to two convergence issues. The first is *energy cutoffs*, which is the cutoff for the wave-function expansion. The second is *number of* $\vec{k}$*-points*, which measures how well your discrete grid has approximated the continuous integral.

The input files

We will look at the `C.scf.in` input file, which is contained in the script that we will use later for the problems. This is an input file for carbon in its diamond form. Diamond is a face-centered cubic structure with two C atoms at 0 0 0 and 0.25 0.25 0.25 (of course, there would be no difference in having those two atoms in 0.13, 0.20, 0.22 and 0.38, 0.45, 047. Why ?). Diamond looks like this (*a* is the lattice parameter).



Look at the input file, which you have just copied over to your directory.

To look at this file, type

```
$ more C.scf.in
```

You can scroll through the file by typing space (to go forward), b (to go backwards) or q (to quit).

The file will look something like this:

```
(1)    &control
(2)        calculation = 'scf'
(3)        restart_mode='from_scratch'
(4)        prefix='diamond'
(5)        tstress = .true.
(6)        tprnfor = .true.
(7)        pseudo_dir = '/state/partition1/marzari'
(8)        outdir = '/state/partition1/marzari'
(9)    /
(10)   &system
(11)       ibrav=  2, celldm(1) =6.60, nat=  2, ntyp= 1
(12)       ecutwfc =40
(13)   /
(14)   &electrons
(15)       diagonalization='david'
(16)       mixing_mode = 'plain'
(17)       mixing_beta = 0.7
(18)       conv_thr =  1.0d-8
(19)       /
(20)   ATOMIC_SPECIES
(21)       C  12.011  C.pz-vbc.UPF
(22)   ATOMIC_POSITIONS
(23)       C 0.00 0.00 0.00
(24)       C 0.25 0.25 0.25
(25)   K_POINTS {automatic}
(26)       4 4 4 0 0 0
```

Lines numbers are added for reference.

Line 1:

`&control` declares the control block.

Line 2:

`calculation = 'scf'` tells PWSCF that this will be a self-consistent field calculation.

Line 3:
`restart_mode = 'from_scratch'`, declares that we will be generating a new structure.

Line 4:
`prefix='diamond',` declares the filename prefix to be used for temporary files.

Line 5:
`tstress = .true.` is a flag to calculate the stresses.

Line 6:
`tprnfor = .true.` is a flag to calculate the forces.

Line 7:
`pseudo_dir = '/state/partition1/marzari',` defines the location of the directory where you store the pseudo-potentials.

Line 8:
`outdir='/state/partition1/marzari'` defines the location of the temporary files. This should always be a local scratch disk so that large I/O operations do not occur across the network.

Line 9:
/ denotes the end of a block.

Lines 10-13: the system block
`ibrav` – gives the crystal system. ibrav=2 is a face-centered cubic structure. This is used because the symmetry of the structure can reduce the number of calculations you need to do. If you need other crystal systems, consult the INPUT_PW file (in ~mounet/LAB2/).

`celldm` – defines the dimensions of the cell. **You will be changing this parameter**. celldm is in atomic units, or bohrs. Remember that **1 bohr** = 0.529177249 **angstroms**. The value will depend on the Bravais lattice of the structure (see below for a key). For FCC, celldm(1) = a_0. In cubic systems, *a*=*b*=*c*= a_0. Consult INPUT_PW for other crystal systems.

`nat` – number of atoms (each individual unique atom). Note that diamond has two unique atoms in the smallest assymetric unit.

`ntyp` – number of types of atoms

`ecutwfc` – Energy cutoff for pseudo-potentials. **This one is important; you will be changing this parameter.**

Lines 14-19: The electrons block

`diagonalization` – diagonalization method. Use the default for now.

`mixing_beta` - Mixing factor. Don't worry about this for now.

`mixing_mode` – Mixing method. Don't worry about this for now.

`conv_thr` – Convergence threshold. Use the default for now.

Lines 20-21: Atomic species declaration
After the keyword ATOMIC_SPECIES, for each ntyp enter
atomic symbol atomic weight pseudo-potential

Lines 22-24: Atomic positions
After the keyword ATOMIC_POSITIONS, for each nat enter
atomic symbol x y z
where x,y,z are given as fractional coordinates of the conventional cell.

Lines 25-26: k-point selection
after the keywork K_POINTS, "automatic" tells PWSCF to automatically generate a k-point grid.
The format of the next line is
nkx nky nkz offx offy offz
where nk* is the number of intervals in a direction and off* is the offset of the origin of the grid.

You can read the documentation for the input file in the INPUT_PW file. This file is in the ~mounet/LAB2/ directory.

The queuing system in hpcbeo2: SGE

The cluster uses a queuing system (here SGE) to manage jobs. Jobs (i.e. scripts containing a set of commands to be executed) are submitted to the queue where they accumulate until some machines of the cluster become free. Then the first job in the queue begins to run into a free machine, and so on.

A few basic commands will be useful to manage your jobs with SGE on hpcbeo2:

- to submit a job to the queue

```
$ qsub C.scf.j
```

This command also gives you the job number on the next line.

- to check the status of your jobs

```
$ qstat
```

- to check the details of one specific job

```
$ qstat -j <jobnumber>
```

where **jobnumber** is the number assigned to your job when you submitted it (you can also get this number by looking at the left column of the output of **qstat**).

- to check the status of all the jobs and all the machines

```
$ qstat -f
```

- to cancel or kill one of your jobs

```
$ qdel <jobnumber>
```

where **jobnumber** is the number assigned to your job.

You can also get the status and use of all the machines on http://hpcbeo2.mit.edu/ganglia/

More information about SGE can be found on the following sites:
http://hpcbeo2.mit.edu/roll-documentation/sge/3.2.0/index.html
http://bio.site.uottawa.ca/wiki/space/How-To/GridEngine/Intro

Running the code using scripts

Both to use the queuing system and to speed up calculations, we will need to use scripts. Following is the general script we will be using for all the calculations.

```
(1)     #!/bin/bash -f
(2)     ##############################################################################
(3)     #$ -N C.scf
(4)     #$ -cwd
(5)     #$ -o $HOME/3.320/LAB2/err.out
(6)     #$ -j y
(7)     #$ -S /bin/sh
(8)
(9)     ###########################################################
(10)    # This is a script to run PWSCF 2.1.2 calculations in
(11)    # the hpcbeo2 cluster.
(12)    # N. Mounet and B. Kozinsky, 03-07-2005
(13)    ###########################################################
(14)    # set the needed variables
(15)
(16)    MYDIR="3.320/LAB2/PROBLEM1/"
(17)
(18)    LISTA='6.6'
(19)    LISTECUT='30 35 40 45 50 60'
(20)    LISTK='4'
(21)
(22)
(23)    USER=`whoami`
(24)
(25)    OUTDIR="/state/partition1/$USER"
(26)    PSEUDO_DIR="/state/partition1/$USER"
(27)    OUTPUT="/home/$USER/$MYDIR"
(28)
(29)    if [ ! -d $OUTDIR ]; then
(30)            mkdir $OUTDIR
(31)    fi
(32)
(33)    if [ ! -d $PSEUDO_DIR ]; then
(34)            mkdir $PSEUDO_DIR
(35)    fi
(36)
(37)    if [ ! -d $OUTPUT ]; then
(38)            echo $MYDIR does not exist, please create it first
(39)            exit
(40)    fi
(41)
(42)    cp /home/mounet/LAB2/C.pz-vbc.UPF $PSEUDO_DIR
(43)    rm -f $OUTDIR/diamond.*
```

```
(44)    ##########################################################
(45)    # calculations
(46)
(47)    for ecut in $LISTECUT
(48)    do
(49)    for k in $LISTK
(50)    do
(51)    for a in $LISTA
(52)    do
(53)
(54)    rm -f $OUTPUT/C.scf.$a.$ecut.$k.in
(55)    cat > $OUTPUT/C.scf.$a.$ecut.$k.in << EOF
(56)     &control
(57)        calculation = 'scf'
(58)        restart_mode='from_scratch'
(59)        prefix='diamond'
(60)        tstress = .true.
(61)        tprnfor = .true.
(62)        outdir = '$OUTDIR'
(63)        pseudo_dir = '$PSEUDO_DIR'
(64)     /
(65)     &system
(66)        ibrav=  2, celldm(1) =$a, nat=  2, ntyp= 1
(67)        ecutwfc =$ecut
(68)     /
(69)     &electrons
(70)        diagonalization='david'
(71)        mixing_mode = 'plain'
(72)        mixing_beta = 0.7
(73)        conv_thr =  1.0d-8
(74)     /
(75)     ATOMIC_SPECIES
(76)      C  12.011  C.pz-vbc.UPF
(77)     ATOMIC_POSITIONS
(78)      C 0.00 0.00 0.00
(79)      C 0.25 0.25 0.25
(80)     K_POINTS {automatic}
(81)        $k $k $k 0 0 0
(82)    EOF
(83)
(84)    rm -f $OUTPUT/C.scf.$a.$ecut.$k.out
(85)    /home/mounet/LAB2/pw.x < $OUTPUT/C.scf.$a.$ecut.$k.in >
   $OUTPUT/C.scf.$a.$ecut.$k.out
```

```
(86)
(87)    done
(88)    done
(89)    done
(90)
(91)    rm -f $OUTDIR/diamond.*  $PSEUDO_DIR/C.pz-vbc.UPF
```

Lines 1-7: These lines are for the queuing system (SGE):

- line 1: shell used for the SGE commands. SGE requires to use the bash shell. **Do not change this line,**
- line 3: the -N option indicates the name to be assigned to the job. You can change it as suits you best (it doesn't really matter how you choose the name),
- line 4: -cwd means to execute the job for the current working directory. **Do not change this line,**
- line 5: -o indicates where the output of the script should go. This is not the output of the PWscf calculations, only the general output of the script, so the file indicated will usually contain only error messages. **You should not change this line,**
- line 6: "-j y" simply tells SGE to put error messages into the same file as the output (the file given in line 5). **You should not change this line,**
- line 7: the -S option tells SGE in which shell is written the script below this line. **You should not change this line** (unless you want to completely modify the rest of the script...).

More information about all these options can be found on
http://hpcbeo2.mit.edu/roll-documentation/sge/3.2.0/index.html
http://bio.site.uottawa.ca/wiki/space/How-To/GridEngine/Intro

Lines 2, 9-14, 44-45: These are just commenting. Any lines preceded by # are ignored by the shell (except the very first line which begins with "#!" and lines beginning with "#$" which are for the queuing system).

Lines 16: "MYDIR" defines the directory where the input and output files will go. **You should change this each time you want your input and output to go somewhere else (e.g. change it to 3.320/LAB2/PROBLEM2/, etc.)**. Do not forget to create by hand this directory (e.g. if you are in the LAB2 directory, you can do mkdir PROBLEM2, etc.).

Lines 18-20: These are where you will change the range of the three main parameters you want to test (lattice parameter *a*, energy cutoff *ecut* and k-point grid *k*, in respectively the variables LISTA, LISTECUT and LISTK). For instance, here we want to test the

variation of the total energy of diamond using different energy cutoffs, so we put in LISTECUT the value '30 35 40 45 50 60', and the code will sequentially calculate the energy for each of the cutoffs 30 Ryd, 35 Ryd, 40 Ryd, etc. **You should change these lines, depending on which parameter you want to test.** But keep in mind that when you put a range for a certain parameter (e.g. here the cutoff), you should keep fixed the two other parameters to a single value (e.g. here we put only one value both in LISTA and LISTK).

Lines 23-43: A few other variables are defined (OUTDIR is the local directory to put temporary files and PSEUDO_DIR is where we put the pseudo-potential file). We also test that the directories exist, and if not, they are created (you can also get an error message if MYDIR doesn't exist – this one should be created by hand). Finally the pseudo-potential file is copied on the local disk, and OUTDIR is cleaned from any previous temporary files found there. **You should not change all these lines.**

Lines 47-52,87-89: This sets up three for-do-done loops. The numbers in LISTECUT, LISTK and LISTA are assigned to the variables ecut, k and a respectively. The word "done" means the end of a loop. **You should not change these lines.**

Lines 54-55: This removes first any previous input file of identical name, and then puts what follows (lines 56-81) into the C.scf.$a.$ecut.$k.in file (where $a, $ecut and $k are replaced by their value) until the EOF is reached (on line 82). Each time the loop is performed, the input is put into a different file, depending on the value of a, ecut and k for the run. **You should not change these lines.**

Lines 56-81: This is just the C.scf.in file, except for line 62-63,66-67 and 81, which contain the variables OUTDIR, PSEUDO_DIR, a, ecut and k respectively. Note that to denote the value of a variable (e.g. ecut), we put a dollar sign in front of it (e.g. $ecut). Remember that here we are changing the cutoff between each run. Every time the script loops through, it substitutes a value for $ecut that it gets from line 19. **You should not change these lines** (except in problem 3 and 4, when testing the forces – you will then change the atomic positions).

Line 84-85: After removing the output file if ever it already exists, this runs pw.x from the input file created above, and gives each output file a different name (in this case C.scf.6.6.30.4.out, C.scf.6.6.35.4.out, etc.).

Line 91: Finally, the node's local directories, where the code copied temporary files, are cleaned.

The script is then submitted to the queue by doing

```
$ qsub C.scf.j
```

It might happen that the queuing system doesn't run the script, and puts it in an "error state" (indicated by "Eqw" when executing qstat). If this occurs, kill it (qdel jobnumber)

and try to submit the script again until it works. If you cannot manage to get it run, ask one of the TAs for assistance.

Organizing your runs

You can organize your runs any way you like, or you don't have to organize them at all. One way is to make directories for each problem you do, and name your output files accordingly. Good organization may save you headache in the long run (but is totally up to you).

Problem 1

Problem 1 will test the energy convergence with respect to energy cutoff.
Look at the script, and look for the parameter *LISTECUT*. Erase everything on this variable after "30", that is, put *LISTECUT='30'*.

Now let's run the script. At the prompt, type

```
$ qsub C.scf.j
```

and wait until the script is submitted and executed (this might take some time) Now, look at the output file

```
$ more C.scf.6.6.30.4.out
```

Scroll through this file. The beginning will just recap the configuration that is being calculated. Then there is some information about the pseudo-potentials that PWSCF just read in. The next part tells you about intermediate energies that PWSCF calculates, before the calculation is converged. Near the end, there will be something like:

```
!    total energy              =   -22.51880584 ryd
```

(your number may be slightly different). Note, the final occurrence of "total energy" will have an exclamation point by it, something you can use to hunt for it. You can skip to this right away by using search functions (in vi, type esc, /total energy, enter), or just scroll down a lot. This is your total energy, as calculated by PWSCF. Or you can type

```
$ grep "! total energy" <filename here>
```

To make sure you have the correct spacings, cut and paste the "! total energy" part from your output file. You can also do this:

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

Which will grep all total energies from all files terminated by ".out" in the current directory. The "!" is a special character, which is negated by using "\".

At the end of the file, it will tell you how long your program took to run, and how much memory it used.

```
     PWSCF        :     0.98s CPU time

     init_run     :     0.14s CPU
     electrons    :     0.81s CPU
     forces       :     0.00s CPU
     stress       :     0.02s CPU

     electrons    :     0.81s CPU
     c_bands      :     0.60s CPU (       6 calls,   0.100 s avg)
     sum_band     :     0.14s CPU (       6 calls,   0.023 s avg)
     v_of_rho     :     0.06s CPU (      13 calls,   0.005 s avg)
  ... etc ...
```

Your numbers may be different from these. You should start to develop a feel for how long your runs take, and how much memory they will use.

```
Cartesian axes

site n.     atom                  positions (a_0 units)
    1           C   tau(  1) = (   0.0000000   0.0000000   0.0000000  )
    2           C   tau(  2) = (   0.2500000   0.2500000   0.2500000  )

number of k points=    8
                  cart. coord. in units 2pi/a_0
   k(   1) = (   0.0000000   0.0000000   0.0000000), wk =   0.0312500
   k(   2) = (  -0.2500000   0.2500000  -0.2500000), wk =   0.2500000
   k(   3) = (   0.5000000  -0.5000000   0.5000000), wk =   0.1250000
   k(   4) = (   0.0000000   0.5000000   0.0000000), wk =   0.1875000
   k(   5) = (   0.7500000  -0.2500000   0.7500000), wk =   0.7500000
   k(   6) = (   0.5000000   0.0000000   0.5000000), wk =   0.3750000
   k(   7) = (   0.0000000  -1.0000000   0.0000000), wk =   0.0937500
   k(   8) = (  -0.5000000  -1.0000000   0.0000000), wk =   0.1875000
```

There will also be lines which say:

```
Cartesian axes

site n.     atom                  positions (a_0 units)
    1           C   tau(  1) = (   0.0000000   0.0000000   0.0000000  )
    2           C   tau(  2) = (   0.2500000   0.2500000   0.2500000  )

number of k points=    8
                  cart. coord. in units 2pi/a_0
   k(   1) = (   0.0000000   0.0000000   0.0000000), wk =   0.0312500
   k(   2) = (  -0.2500000   0.2500000  -0.2500000), wk =   0.2500000
   k(   3) = (   0.5000000  -0.5000000   0.5000000), wk =   0.1250000
   k(   4) = (   0.0000000   0.5000000   0.0000000), wk =   0.1875000
   k(   5) = (   0.7500000  -0.2500000   0.7500000), wk =   0.7500000
   k(   6) = (   0.5000000   0.0000000   0.5000000), wk =   0.3750000
   k(   7) = (   0.0000000  -1.0000000   0.0000000), wk =   0.0937500
   k(   8) = (  -0.5000000  -1.0000000   0.0000000), wk =   0.1875000
```

This records the number of unique k-points that were calculated. The input here has a 4x4x4=64 k-point mesh, however some of these k-points have the same energy because of the symmetry of the crystal. They are then weighted differently. The weights add up to 2, an idiosyncrasy of this program. Your numbers will be different if you use a different k-point mesh

Now, increase the cutoff in the C.scf.j file, and rerun the calculation. A good increase in cutoff would be ~10 ryd.
To finish problem 1, put a list of different cutoffs in LISTECUT and check all the resulting energies.

Problem 2

Problem 2 will test the convergence of the energy with increasing the density of the $\vec{k}$-point grid. We will be testing $\vec{k}$-point grid convergence and cutoff convergence separately. So, set your cutoff to something lower, such as 30 Ryd (or some other cutoff that you have used already in Problem 1). If you have saved all of your output files, there should be no need to rerun an initial calculation. There are some "cross effects" in testing cutoff and $\vec{k}$-point separately, however we assume these are small.

To increase the size of the $\vec{k}$-point grid, edit the script and change the values in *LISTK*: put '6 8' instead of '4'. Resubmit the script, and record the total energies for the two grids 6x6x6 and 8x8x8. You may also want to test denser grids.

You will also want to record the number of unique $\vec{k}$-points (that is, unique by symmetry). This is near the beginning, and looks something like this:

```
Cartesian axes

     site n.     atom                  positions (a_0 units)
         1        C     tau( 1) = (  0.0000  0.0000  0.0000  )
         2        C     tau( 2) = (  0.2500  0.2500  0.2500  )

     number of k points=   8
```

Your calculation will scale roughly as the number of unique $\vec{k}$-points. You can verify this with the timing information.

Problem 3 and Problem 4

In problems 1 and 2, the forces on C are 0 in the x, y, and z directions. This is because of symmetry, which cancels out forces. In problems 3 and 4, we will create forces by displacing a C atom +0.05 (fractional) in the z direction. To do this, edit z coordinate of the C ion in the input file. After you edit the script, the ATOMIC_POSITIONS section will look something like this:

```
ATOMIC_POSITIONS
 C 0.00 0.00 0.00
 C 0.25 0.25 0.30
```

Now put '30' in *LISTECUT* so that the cutoff is 30 Ryd, and put '4' in *LISTK* so that the $\vec{k}$-point grid is 4x4x4. Rerun the script, and record the forces. The forces will appear in the output file after the total energies. It will look something like this:

```
Forces acting on atoms (Ry/au):

atom   1 type  1   force =     0.00000000    0.00000000    0.25834978
atom   2 type  1   force =     0.00000000    0.00000000   -0.25834978

Total force =    0.365362      Total SCF correction =     0.000005
```

The numerical value for your forces may be slightly different from the above example. Forces are given in units of Ryd/bohr. Record this number, and retest the convergence issues with respect to cutoffs and $\vec{k}$-points.

Problem 6 and Problem 7

Given the information you learned above, you should be able to do problem 6 and 7. For problem 7, put only one value in both *LISTK* and *LISTECUT* (the converged values), and a range of values in *LISTA*. Good luck!

FAQ for HW 2

I do not understand quantum mechanics at all.

General introductions to Quantum Mechanics
http://walet.phy.umist.ac.uk/QM/LectureNotes/QM.html

In particular look at Chapters 1, 2, 3, 8, 11.

Operators, expectation values, bra-kets: Paragraphs 1.1 and 1.2 of
http://www-keeler.ch.cam.ac.uk/lectures/quant_letter.pdf

Or, more complete:
http://www.math.utah.edu/~gold/doc/quantum.pdf

Very simple primer:
http://www.chem1.com/acad/webtut/atomic/

Note, it is not always important to know every detail of quantum mechanics to run a quantum mechanics code.

How precisely do I need to get the lattice parameter?

Lattice parameters are typically listed to within 0.01 Angstroms.
There are applications when higher precision is required; this is not one of them.

The energies when you move an atom (the force calculations) are higher than when you don't?

This is correct. Remember equilibrium has the lowest energy. Equilibrium for this structure has C at 0 0 0 and at .25 .25 .25. As a side note the forces will give you an idea of how far you are from equilibrium (they tell you which direction the atoms "want" to move). The stresses tell you which direction the cell parameters "want" to change to reach equilibrium.

My E vs. lattice constant plot is jagged.

There are a number of solutions to this; the easiest is to raise the energy cutoff.

I don't like scripts.

Use scripts! They will save you a lot of time in the long run. This is anyway the only way to use the queuing system.

How do I kill my script?

Type qstat. This will tell you which jobs you are running – look at the numbers on the left. Type "qdel <number here>" to kill your job.

The weights of the k-points add up to 2, not 1.

Yes. This is a "feature" of the code. Don't worry about it.

How is "convergence of energy" defined?

You say that your energy is converged to X Rydbergs when E_{true}-E_n=X (n is the current energy). How do you know E_{true}? In practice you might take your energy at the highest cutoff (or k-grid, or whatever) that you calculated – if that seems converged, you might call that E_{true}. The most important thing is that you do NOT define convergence as E_{n+1}-E_n, where n is a step in k-point, energy cutoff, or whatever.

You do need to be careful though. It is possible to get "false" or "accidental" convergence as well. That is, your energy at a 2x2x2 k-grid may be the same as the energy at a 8x8x8 k-grid, but the energy at a 4x4x4 might be very different from both of these. In this case, you aren't really converged at a 2x2x2 k-grid.

I don't understand convergence of energy and forces. It seems that, as a percentage of the absolute value, energies converge much faster.

Sometimes you are interested in an absolute value, rather than a percentage value. For instance, let's say you can measure the length to within 1mm. If you measure the length of an ant, in terms of percentage error, you may be off by 50% or more. If you measure the length of an elephant, in terms of absolute error, you may be off by 0.01%. But usually you don't care as long as you are to within 1mm.

Errors on forces are the same. Don't worry about the percentage errors so much. You could always arbitrarily decrease the percentage errors on the forces by taking a bigger displacement.

From experience, we know that a good error on energy differences is ~5 meV/atom.
From experience, we also know that a good error on forces is 10 meV/Angstrom.
These are just values that we know, because we have done many first-principles calculations in the past.

This is what you should look for.

I am having problems both converting Ryd to eV and converting bohr to Angstroms

The units page may help you.
http://www.chemie.fu-berlin.de/chemistry/general/units_en.html

Does PWSCF use LDA or GGA? DFT or Hartree Fock?

PWSCF uses DFT. It has both LDA and GGA, but in this lab we only use LDA.

My energies are really different from lab 1. What is the 'correct' scale I should be looking for?

Remember that absolute energies do not usually mean very much. We are mostly interested in energy differences. Also, the reference energies are different. In lab 1, the reference energies (when atoms are infinitely far apart) is 0. In lab 2, this is not the case. The absolute energies can shift a lot, depending on which reference energies you take.

Why do I take symmetric k-point grids? Can I take asymmetric k-point grids?

For this material, we take a symmetric k-point grid because it is the same in all three lattice directions. This is not always true. The 'best' k-point meshes are those that sample k-space evenly in all directions. Thus, for a tetragonal cell (with a=b, c=2a, and all angles 90 degrees) we might take a 8x8x4 mesh. Think about why this is (Note that if a lattice vector is longer in *real* space, it is shorter in *k*-space).