Brief introduction to Abinit
Short lecture notes for the IDEA league summer school 2007

Pierre-Matthieu Anglade

July 30, 2007

Contents

Chapter 1

Introduction

Being able to perform multiscale calculations is one of the major goal of current simulation development effort. It is very desirable to be able to ground multiscale simulations on a firm as rock basis at the atomistic level. This firm base is provided through the well known Schrödinger equation. However exact solution for non trivial problem (counter e.g.: isolated hydrogen atom or homogeneous electron gas) of this equation are hardly derivable; especially on paper. That's why a large effort has been devoted during the previous century to develop approximation allowing to derive accurate solutions for problem of scales and interest much larger than the above cited counter examples while still quite small even compared to the smallest ink droplet of ink-jet printers.

It is not the topic of those notes to describe the history of so called abinitio methods. Let just state that currently development are still trying to increase both accuracy of calculations and the size of systems within reach. This last goal is mainly pursued through the development of order N algorithms. Examples of codes achieving this are, to name but a few, Conquest (http:www.conquest.ucl.ac.uk) and BigDFT (http:www-drfmc.cea.fr-sp2mL_SimBigDFTindex.en.html). Presently, abinitio methods, using the local density approximation (LDA) or the generalized gradient approximation (GGA) together with pseudo-potentials are quite successful at reproducing and predicting accurately lots of materials properties.

Many codes exist that uses those approximations. One of them is Abinit. This code can draw attention by lots of aspects, among which it's availability under the GPL license is not the least. It's ease of use was one of the main factor that lead to present it at this summer school. Moreover, probably be-

cause of its development process and licensing scheme, Abinit has a very large documentation that allows easy learning. Most informations not included in the present notes about Abinit can be found at www.abinit.org. This includes many tutorials, keywords documentation and complete commented source code.

The goal of the present lecture notes is multi folds. First they aim at introducing briefly Abinit's package, community, and development model to the reader. Second they must be an introduction to basic DFT calculations. This is achieved through the example of Abinit's own input structure. In this part we are going to emphasize the very minimal technical knowledge one must have to perform DFT calculations. As a conclusion to this second chapter we propose you a simple calculation project which will help you evaluate the possibilities and pitfalls of DFT calculations. Third the lecture notes shall introduce the reader to the various calculation possibility available with Abinit. Interested student will then be able to go through the tutorials to get deeper technical knowledge on those advanced features.

1.1 Abinit's basics

1.1.1 License and consequences

In order to favor cooperation Abinit is delivered under the GNU general public license that offers very interesting guarantees for scientific collaborations. Among them is the fact that you can not be deprived of your own work or of your tool since the code is really free and guaranteed to stay so.

By the way, favoring cooperation has some nice side effects. Among them is the necessity to have a fairly well documented code. You can find all the documentation in Abinit repertory, in the sub-directory "doc" or most of it on-line at abinit.org. This documentation helps both beginners and developers; both peoples wanting to use the code and people wanting to do their own modifications.

Another consequence of the GPL is the presence of a fairly large community of users and developers around Abinit. For instance, a tangible consequence of this is the very short response time on Abinit's forum where people asks for help.

Third, side effect is that Abinit is a reliable, peer-reviewed scientific code. Many groups modify Abinit. They need to read and check modifications by

others. Moreover X. Gonze and coworkers have had to introduce robust automatic quality checking of the code to enable the large worldwide cooperation that Abinit has become.

1.1.2 Code (building, etc.)

Abinit package is composed of it's main program Abinit but also of many other code. Some of them have only internal uses and others are almost independent tools. If you delve in the codes you'll find languages from bash to fortran, from Perl to C passing by python. Yet even developers needs to know only fortran to work with Abinit.

In fact compiling all the code is pretty straight forward even for absolute beginners. Abinit 5.1 and later build system is based on the *de facto* standard of autotools. So as long as a fortran 90 compiler is in your PATH you can just go in Abinit directory (for instance after downloading and untaring Abinit's package) and type `./configure; make`[1]. After a while Abinit and all binaries are now build. You will find them in `src/main/`.

Fortran 90 being what it is, even the best code can fail because of evil compiler. Even the best compiler can fail because of spurious code. Do not trust compilation results without checking them. You must go into the test directory and run the tests. Afterward you'll be able to judge if the compilation runs fine. To do this just type: `cd tests; make tests_min; nedit */*/report`.

Report files contains lists of tests and comments about them failing or succeeding. It seldom happens that no tests fails. However you are now aware of what can be done or not.

For instance if you happen to debug a failing tests (beware that problems are usually more about compilers than about Abinit's code) or to develop a useful function feel free to contribute it by sending a patch to professor X. Gonze by email (gonze@pcpm.ucl.ac.be).

[1] or alternatively: `mkdir tmp; cd tmp; ../configure; make`

Chapter 2

DFT calculations basics

DFT calculations require usage of multiples algorithms and numerical methods. Each of them may require some tuning in order to output accurate results. New users may hardly have all of them in mind. That's why it is recommended to start new calculations by tuning old input files. You will find hundreds of such examples in Abinit tests directory.

2.1 Abinit input files

Lets start our introduction by explaining a typical sample of input file for Abinit presented in figure 2.1. This file is mainly divided into four parts: Starting at line 2 is the section that is going to control the quality of the integration of energy over bands in reciprocal space. At line 12 begins the definition of the unit cell. The third section, from line 23 to 25, commands the precision of the plane-wave basis set. Finally, at line 26 starts the part which describes the self-consistent field cycles algorithm and convergence. We will explain all this in detail below.

Cell geometry

First let's start with ions positions and cell geometry. At line 13 we specify the length of the three basis vector. The default unit used by Abinit is bohr, that is atomic units (1 bohr $\simeq$ 0.529 Å). We specify twice 5.037618 and once 9.351229. Axis a and b will then have the same length of almost 2.5 Å, while axis c will measure about 5 Å. It is possible to specify this by writing (for instance):

```
1  ndtset 14
2  #Reciprocal space integration
3  kptopt 1
4  nshiftk 1
5  shiftk
6  0.0 0.0 0.5
7  ngkpt 4 4 4
8  tsmear  0.001
9  occopt 4
10 #use wavefunctions from previous dtset
11 getwfk -1
12 #Definition of the unit cell
13 acell  2*5.037618  9.351229
14 angdeg 90 90 120
15 ntypat 1
16 znucl 30
17 nband 18
18 natom 2
19 typat 2*1
20 xred
21   0.0  0.0  0.0
22   1/3  1/3  1/2
23 #Definition of the planewave basis set
24 ecut: 5
25 ecut*   1.414213562373095
26 #Definition of the SCF procedure
27 nstep 25
28 toldfe 1.0d-7
29 iprcel 145
30 iscf   7
```

Figure 2.1: Input file used to get the evolution of error with respect to cutoff energy in the case of zinc pseudo-potential with twelve electrons. Note that line numbers are NOT part of the input file.

```
acell 5.037618 5.037618 9.351229
```

or

```
acell 2.5 2.5 5 angstroms
```

Note that using the `2*` notation used in figure 2.1 is especially convenient since very brief. This may be very useful in case of large arrays. The parameter `angdeg` at line 14 specify the angles in degrees between axis. First angle between a and c, second between b and c and then between a and b. The keyword `angdeg` is mostly used when we want to input hexagonal lattices. It assure that the angles between axis is exactly what it is meant to be. Alternatively one can also specify the matrix of primitive vectors in the form

```
rprim
a1 a2 a3
b1 b2 b3
c1 c2 c3
```

Where ax designed the x coordinate of axis a. This matrix is going to be multiplied by the vector `acell` to define the actual simulation cell. In our case, for an hexagonal lattice, we can write:

```
rprim
 1   0   0
-0.5 0.8660254037844386  0
 0   0   1
```

The number in the middle must be written with care and lot of decimal in order to get exactly the geometry described previously with angdeg. This is not very convenient and is the occasion to introduce an other trick of Abinit's input file. You can write the following:

```
rprim
 1   0           0
-0.5 sqrt(0.75)  0
 0   0           1
```

where $\sqrt{0.75}$ will be evaluated precisely at run time.

At line 15, `ntypat` specify the number of atomic species in the simulation. then follows an array describing the atomic number of the present species introduced by `znucl`. Subsequently every time atom species matters they will be exactly in the order specified in `znucl`. Here `znucl` is obviously a single number since we have a single species.

The keyword `nband` is critical. It describes the number of wave functions (bands in term of solid state physics) which are going to be explicitly included in the calculation. That is we must have enough to hold all our electrons. For instance for a non spin polarized calculation (that is when considering that 2 electrons can occupy the same orbital) the number of bands must be greater or equal half the number of electron. For non conducting solid half the number of electron is OK. Yet for metals where electrons spreads into conduction bands we may need to add a lot of bands. After the calculation it is always recommended to check the occupancy of bands in the output file. The last band included must be empty or almost empty to ensure a correct result.

With `natom` we specify the number of atoms in our cell. Even though obvious in our case it is mandatory to specify for each atom its type. This is done with `typat` at line 19 where "1" means atom of the first type specified in `znucl`, "2"... Again for this array it may be useful to use the shortcut $m*n$ meaning that element n is repeated m times.

Specifying atomic positions can be done with several keywords: `xangst`, `xcart`, or `xred`. The first one allows definitions in angströms, the second in default real-space unit (bohrs), both in Cartesian coordinates and the last one means numbers will be given in reduced coordinates (along vectors of `rprim`). This method is very convenient for crystal structures. Here we used the usual reduced coordinates for atoms in hexagonal closed pack structures. Note the way we can specify fractional number as quotient. Again this is quite useful when dealing with rational numbers.

Reciprocal space integration

Integration in reciprocal space are performed through classical integration mechanisms. That is a mesh of point where functions (wave functions energy) are evaluated must be defined. Within Abinit one usually do this with the 5 input variables `kptopt`, `nshiftk`, `shiftk`, and `ngkpt` or `kptrlatt`. The game consist in finding the most homogeneous way to sample reciprocal space. Sampling points are called k-points.

First we set `kptopt` to 1. This is the usual value for ground state calculations. It means that we use an automatic array of k-points with every symmetry. You must confer to the Abinit documentation for more information about options you have.

Then `nshiftk` and `shiftk` define origins of grids. Using `nshiftk`=0 or

`shiftk 0 0 0` means that the grid start at the Γ point. For instance one can use a ccp grid by specifying

```
nshiftk 4
shiftk
0.5 0.5 0.5
0.0 0.5 0.5
0.5 0.0 0.5
0.5 0.5 0.0
```

which is usually highly efficient for cubic cells. In the case of figure 2.1. Note that it is almost never a good idea to sample the Γ point: It's special properties makes calculations last longer. That's why one uses at least one shift for usual calculations.

Two variables, `ngkpt` and `kptrlatt` are used to define the meshes. `Ngkpt` allow to define the usual, so called Monkhorst-pack, k-point mesh. It corresponds to defining the diagonal elements of the `kptrlatt` matrix which in turn is made of three real-space vectors whose coordinate are expressed in the `rprim` basis. K-points will occupy nodes of this super lattice. The quality of the mesh can be characterized by the smaller distances between nodes which is called `kptrlen` within Abinit.

A fine grid helps performing a precise integration. However, for metals, the required precision of convergency is usually too difficult to reach at 0 K. Then one uses an occupation function defined by the variable `occopt` associated with a finite electronic temperature. Together those parameters will smooth the fermi surface. Since the final goal is smoothing more than the introduction of a physical temperature, various occupation functionals are available. In general the cold-smearing developed by N. Marzari (`occopt 4`) is OK for minimizing the thermal effect while smoothing the electron distribution. The electronic temperature (by default in atomic energy units, that is hartree) is defined by `tsmear`. One must remember that 1 mHa $\simeq$ 300 K. Making usual smearing temperatures of 0.01 Ha very high.

Plane wave basis set

Plane waves are a systematic basis set. Definition of the basis is then very easy and can be controlled by a single parameter which is the kinetic energy cutoff. This parameter is known as `ecut` within Abinit. It usually vary between 5 for the smoother pseudo-potentials to about 100 Ha for the steepest.

You can see that this keyword appears twice in the input file. This is to demonstrate an other feature of Abinit: the multi data-set mode. It is possible to chain calculations within one program run. This is especially convenient when we need to do many related calculations. For instance, the input file of figure 2.1 is intending to figure out the convergence of calculations with respect to `ecut`. At line 1 we define the number of chained calculations. All of them will use the same value for the parameters as specified in the input file, except for `ecut` due to the special syntax chosen. The columns at the end of `ecut` at line 24 specify that we define here the starting value for `ecut` that is it's value for the first data-set. It also tells Abinit that `ecut` evolution through data-set will be a suite. At line 25 the star (`*`) tells Abinit that the suite is a geometric sequence. Line 25 also define the common ration to be $\sqrt{2}$.

The fact that we are going to chain multiple calculations appears also at line 1 and 11. Keyword `ndtset` specify the number of calculations. For each of them all parameters will be kept constant (as specified in the input file) unless some special instructions are given (e.g. `ecut`). One can override the default for a specific data-set by specifying its number. For instance if we had `tsmear13  0.01` in the input file the electronic temperature will be increased to 0.01 hartree for the thirteen data-set only.

At line 11 the `getwfk -1` require that Abinit takes the previously computed wave functions as a starting point for each data-set. This will increase considerably calculation speed since we are always considering the same system with just more and more plane-waves. This requires that Abinit writes the wave functions of each systems. Requirement which is fulfilled because, has you can see in documentation, the default value for the `prtwfk` (print wave-function) parameter is one.

Self consistent field cycles

One of the day to day challenges of peoples doing DFT calculations is to get the convergence of the so called self-consistent field (SCF) cycles. To obtain that we must first define conveniently what convergency means. And then decides on the algorithms which will solve problem.

Many criterion can be used to define convergency and the convergency requirement for different problems may be very different. For instance when doing phonon calculations you will need to know the wave functions and potential within machine precision. At the same time if you want to get a

correct dielectric matrix convergence of total energy within 10^{-4} hartree is sufficient. Sometimes it is convenient to get the convergency of forces. Yet this quantity is absolutely meaningless in cells where symmetries make all forces vanish. In the case of figure 2.1, because our interest is on total energy, and because we have no special need for a high level of convergency we use the keyword `toldfe`. It will make Abinit stop the SCF cycles when the changes in total energy will be smaller than the defined value twice in a row. We also specify that whatever happens the SCF loop will end after, at most, 25 steps. This is done by writing `nstep 25`.

In the present case reaching self-consistency is not challenging. Default value would have done the trick. Input variables that controls self-consistency within Abinit are mainly `iscf`, `iprcel`, and `diemac`. `Iscf` controls the so called mixing scheme, that is the minimization algorithm which leads to convergence. The default value of 7 chooses the best algorithm that is Pulay's mixing.

`Iprcel` controls the preconditioner; that is an operator in charge of simplifying the self consistent problem. There exist many such operators of which the one refers to by `iprcel=4x` is probably the most efficient and multipurposes. Note that for our homogeneous cells (no vacuum) it is much faster and almost as efficient to use `iprcel=0` which selects Kerker's preconditioner. The key parameter for this preconditioner is `diemac` which shall be chosen proportionally to the problem toughness also known (almost) as the macroscopic dielectric constant. This last quantity evolves as the square of super cells size for metals and is almost constant for non metals.

2.1.1 VIV: very important variables

Abinit usually set safely all of its variables however it is very likely that once or then you will have to use one of the following:`ixc`, `nnsclo`, and `nline`.

The first one is used to select the exchange and correlation function. You have a large choice between lots of LDA, GGA and other functions. If all your pseudo-potential use the same exchange and correlation functional, Abinit will use this one. Otherwise it will use `ixc=1`. If you want to do really abinitio calculation the choice of Abinit is correct as long as all your pseudo-potential uses the same exchange and correlation function. Otherwise you may want to tune this function to best reproduce some known properties of special interest for you.

The convergence of the density to the ground state should be an bal-

anced process between the convergence of wave functions and of the density/potential itself. Yet sometimes, for very large system or for system with large energy cutoff, the settings of the conjugate gradient which deals with wave functions optimization are too conservative to get a nice convergency. The algorithm is controlled by the two variables `nnsclo` and `nline` The first of them controls the number of restart per SCF step of the CG. The second the number of line minimization to be performed per restart. Later we will learn how to diagnose a bad convergence of wave functions. You need to remember that restarting a conjugate gradient is useful for an-harmonic functions (far for convergency) while increasing the number of line minimization is the usual and most efficient procedure for harmonic ones.

2.2 Running the code

Once compiled the binary file *abinis* stand in you `src/main/` directory. If you've build the parallel version of Abinit you have also a binary called *abinip*. The first can be used directly. The second needs to be launched by your mpi script. In this section we deal with what is happening after starting the run.

2.2.1 Input

Abinit is user friendly and will politely ask you for a number of information. In order they are:

- name of your input file,
- name of an output file,
- prefix for other (called generic) input files,
- prefix for other (called generic) output file,
- prefix for temporary files,
- and finally one pseudo-potential file for each atomic species specified in your input.

First let's notice that you must answer to each of this question for each run. This is a bit heavy but you can skip this step by writing your answers in a file. Let's call this file `answers.files`; it may look the following:

```
a.in
a.out
a_in
a_out
/tmp/a_tmp
pseudopotential_file1
pseudopotential_file2
...
```

Now we can start Abinit by the following command:
`/path/to/abinis < answers.files`.
Under unices shells this redirect the so called standard input to our file `answers.files`. After launch, Abinit starts the calculation and writes a lot of comments and details to the standard output. Again we usually redirect all this to a log file.

Abinit will also write an output file named after the second line of `answers.files`. If this file already exists, Abinit append a capital A to the name. If it exists also it tries a B and so on. If all files exists until Z, you'll get an error message.

Generic input files are used through variables like `irdwfk` during the run. For instance if you want to read the wave functions written during run A in subsequent run B, the prefix of generic input file of run B must match the prefix of generic output file of run A.

Generic temporary files can usually be forgotten about. That's why they are directly send to tmp in the example above.

2.2.2 Output

Abinit output file is a (long) structured ascii file in which you must be able to find important information such as calculations results (always useful right?).

It is divided in three sections. First it reminds you every information you have given to Abinit. Telling you some details about the various resulting calculation parameters. For instance for each data-set you get a resume looking like this:

```
 DATASET    7 : space group Pm m a (# 51); Bravais oP (primitive ortho.)
================================================================================
```

```
 Values of the parameters that define the memory need for DATASET  7.
   intxc =         0  ionmov =         0    iscf =         7     ixc =         1
  lmnmax =         6   lnmax =         6   mband =         8  mffmem =         1
P  mgfft =        54   mkmem =        14 mpssoang=         3     mpw =      2504
  mqgrid =      3001   natom =         2    nfft =     48600    nkpt =        14
  nloalg =         4  nspden =         1 nspinor =         1  nsppol =         1
    nsym =         8  n1xccc =         0  ntypat =         1  occopt =         4
 For the susceptibility and dielectric matrices, or tddft :
   mgfft =        30  nbnd_in_blk=     4    nfft =      9720     npw =        37
================================================================================
P This job should need less than                      20.316 Mbytes of memory.
  Rough estimation (10% accuracy) of disk space for files :
  WF disk file :      4.281 Mbytes ; DEN or POT disk file :      0.373 Mbytes.
================================================================================
```

This is written at the time Abinit is checking your input. You get the value of some important variables. Whose meaning can be checked in Abinit documentation. The remaining of important input variables are echoed afterward:

```
     natom          2
     nband          8
    ndtset         14
     ngfft1        12      12      20
     ngfft2        15      15      24
     ngfft3        16      16      30
     ngfft4        18      18      32
     ngfft5        24      24      40
```

You can see that the syntax is close to that of the input file. Variables that are kept constant along data-sets are written once. Changing variables are written with an appended number. For instance, here the number of grid point used to describe the function and to perform Fourier transforms in each direction (`ngfft`) changes between data-sets. This is a direct consequence to the increase of `ecut`.

The second section of the output file describes evolution of critical quantities along SCF cycles.

```
================================================================================

     iter   Etot(hartree)      deltaE(h)  residm     vres2    diffor    maxfor
 ETOT  1  -2.2196893333886    -2.220E+00 7.148E-03 5.059E-01 6.147E-03 6.147E-03
 ETOT  2  -2.2201091159529    -4.198E-04 3.542E-05 1.971E-03 1.143E-04 6.032E-03
```

```
 ETOT  3  -2.2201101669211    -1.051E-06 2.908E-05 5.203E-05 5.082E-05 5.981E-03
 ETOT  4  -2.2201102191573    -5.224E-08 1.518E-05 6.066E-06 6.539E-06 5.975E-03
 ETOT  5  -2.2201102267255    -7.568E-09 4.939E-05 2.273E-08 6.160E-07 5.974E-03

 At SCF step    5, etot is converged :
  for the second time, diff in etot=  7.568E-09 < toldfe=  1.000E-07

 Cartesian components of stress tensor (hartree/bohr^3)
  sigma(1 1)=  7.07589342E-04  sigma(3 2)=  0.00000000E+00
  sigma(2 2)=  9.69369244E-04  sigma(3 1)=  0.00000000E+00
  sigma(3 3)=  7.94392636E-04  sigma(2 1)=  2.26708045E-04

================================================================================
```

For each cycle, one line is written. It starts with `ETOT` and the number of the iteration. Then comes the total energy found (`Etot`) and the change in total energy with respect to the previous iteration (`deltaE`). `Residm` is the max residual of wave functions ($residm = \max(< \psi_i|(H - \epsilon_i)^2|\psi_i >)$) It tells you the level of convergence for the worst converged wave function during the current iteration; yet it is not significant of the overall convergency.

One of the most significant outlier of convergency is the average of square potential residuals `vres2`. Let V_{in} be the potential at the start of one SCF cycle and V_{out} the one at the exit resulting from optimization of energy with respect to wave functions in the guessed Hamiltonian. The residuals are just $V_{res} = V_{out} - V_{in}$. Thus at convergency $V_{res} = 0$ making $E(V_{res}^2)$ a nice convergence criterion.

For molecular dynamics or cell optimization we do not need absolute convergency but only accurate forces. Then the maximum difference between forces at the previous iteration and forces at the current one (`diffor`) can be a convergence criterion.

The third part of the output file is a resume of all important quantities at the end of the calculation. There again you will find all important quantities. The most important thing to check is occupation numbers of the last band (wave function) at each k-points it must be zero or very close to zero. Otherwise it is a sign that you have not included enough bands into your computation.

2.2.3 Log

The log file is mostly an expanded version of the output file. It contains lots of comments and warning. That's why it is usually saved. It may be helpful to diagnose problems.

2.3 Example of brass 50% study

2.3.1 Introduction

The example of brass will make a fine example for a typical DFT study. It allows to introduce most typical difficulties while not presenting a tough problem. Indeed brass 50% is a body centered crystal alternating zinc and copper. This special property will ease greatly our calculations: not only unit cell for brass 50% is smaller than that of any disordered alloy but it also retains much more symmetries.[1].

2.3.2 Pseudo-potentials

The first step in a plane-wave ab initio calculations is to choose pseudo-potentials. And to evaluate their properties. We chose to use pseudo-potentials of the HGH (Hartwigsen, Goedecker, Hutter) formalism here. There exists two for both elements. One with only s electron and one with both s and d electrons.

Evaluating the properties of a pseudo-potential is a twofold task. One needs to be able to control the accuracy of the various approximations used to solve the DFT problem; and then it is mandatory to check the ability of the pseudo-potential at describing some well known properties of the system under investigation.

In the usual case, for metals, the first step consists in determining the effects of three parameters: the plane-wave basis cutoff energy, the sampling of the first Brillouin's zone (FBZ) and the electronic temperature. Fortunately

[1]Readers must be aware that abinitio codes usually takes care of symmetries to accelerate calculations and increase precision. The drawback of this is that no natural fluctuation is going to break algorithm enforced symmetries. This sometimes makes molecular dynamic results from a DFT code very different from their classical counterpart. However if needed you can enforce symmetry breaking within Abinit.

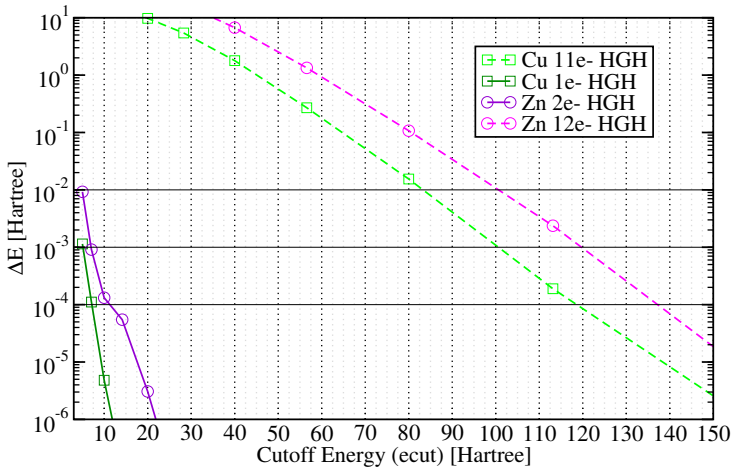

Figure 2.2: Energy convergence as a function of the plane-wave cutoff energy for the four HGH pseudo-potentials. The squares are point calculated for copper; circles stands for zinc results. Continuous line draws the evolution of convergence for the "s-only" pseudo-potentials while dashed line represents the evolution for the "sd" pseudization. The three black horizontal line label convergence level of special interest: at a level of 10^{-2} hartree the convergence is never enough for a true system study; yet it is enough to hold reasonable school results; 10^{-3} hartree is typically the level of convergence used in pseudo-potentials abinitio studies because it is also the usual level of convergence between a pseudo-potential and the full-electron calculation used to generate it; finally 10^{-4} hartree is a level of convergence seldom used when one want a very precise calculation.

the cutoff energy is almost independent of the two other parameters. Monitoring the level of accuracy consist in checking that the difference in total energy for a converged calculation is below a certain level when compared to the same calculation with an absolutely converged value of the parameter under investigation.

Cutoff energy

The first approximation to be checked is the extension of the plane-wave basis set required to get an accurate description of the system. This number depends solely on the pseudo-potential. Within Abinit it is controlled by the

parameter `ecut` which is the maximal kinetic energy of the plane-waves used to describe the wave functions.

Because we will need to know whether our pseudo-potentials are correct or not we can't start working with the BCC brass. In a different world where we would knew for sure (from previous experiences for instance) that the pseudo-potential are accurate it would be quite possible to check directly in brass the correct value for ecut. We would have obtained the biggest cutoff energy of those required for Zn or Cu. Here however we must get each of those cutoff energy.

Using input files like the one displayed in figure 2.1 we get total energies corresponding to each value of the cutoff energy. They can be compared to the most converged of them to get figure 2.2.

In figure 2.2 it is striking that the pseudo-potential with less electrons require much smaller values of ecut. Both materials seems to behave exactly the same way. This is quite awaited. The deeper the electrons the higher the cutoff must be. This is precisely the reason that as leaded to the development of pseudo-potentials.

Accurate sampling of the Brillouin's zone

In periodic systems calculations, the total energy of the system is obtained through integration of bands energy across the FBZ. It is then mandatory to sample accurately this area. This is done through what is called a *k-point* mesh. This is a very usual integration problem. The precision is then dependent upon usual parameters. The steeper the bands variations the more k-points are required. The largest the cell (the smaller the FBZ), the less k-points are needed.

One of the most crucial question is to know if any band crosses the fermi level. That is to say if we are considering something with a metallic behavior or an insulator. In the first case it is quite often mandatory to have a very accurate sampling of the FBZ, because bands are populated only up to this level which then shall be almost perfectly described. Conversely, materials with a gap can be described with loose k-point meshes.

Usually for metals (or any material described as metallic in DFT) one will have to use a very large number of k-points. That may be numerically untractable. Alternatively, instead of increasing the accuracy of the mesh, one can make the problem easier by smoothing the borders of the fermi surface. In fact this is the natural effect of electronic temperature. It spreads

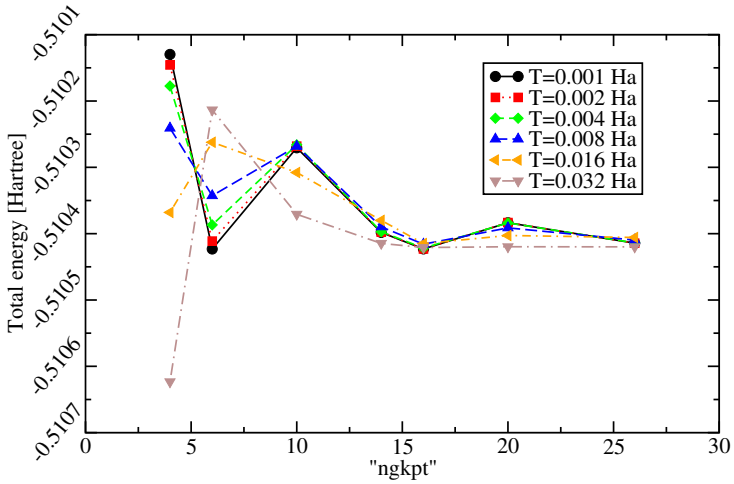

Figure 2.3: Energy with respect to monkhorst pack k-point mesh for single atom CCP copper cell with the single electron HGH potential. Each couple line/symbol represent energy evolution for electronic temperature ranging from 1 mHa to 32 mHa. The smearing function used is that of N. Marzari.

the population of electrons making integration across the fermi surface much easier. Within Abinit the electronic temperature is defined via the parameter *tsmear*. Nevertheless the fermi distribution of electrons is not the best suited functions to reach smooth integration. Indeed, we try to compute ground state properties of materials. Then what we want is to get the smoother integration with the smallest perturbation of the wave-functions. This can be reached by so called cold-smearing functions as developed by N. Marzari. To select between various smearing functions one use the variable *occopt*.

For non metallic systems the convergence of the k-point mesh is a simple task that can be performed directly just in the way demonstrated for ecut previously. For metals we must conduct a two variable optimization. It will result in a set of (temperature,k-mesh) that achieves our convergence criterion. We then have to discriminate on the basis of equilibration of, on the one hand, physical properties, on the other hand computational cost.

In our metallic case, at the starting point we need to evaluate the combined effect of k-mesh and smearing. The result is displayed for copper in figure 2.3.

First, one notice that energy is not variational with respect to k-point

sampling. To evaluate the degree of convergence of a particular mesh we must then take into account not only its own energy difference with respect to the converged case but also that of its neighbor. For instance if we look at the second point for the lowest temperature case. Its difference with the converged case is fairly minimal. This happens accidentally by error cancellation and is a testimony of the need to consider neighboring meshes errors to determine the level of convergence obtained.

Second, as expected, it appears that convergence for a given mesh is better for higher temperatures. They should allow the use of smaller k-point meshes. In our particular case it will not matter tremendously. However this is of uttermost importance for lots of metals.

Third, we note that even for the smallest k-meshes the energy is converged better than 1 mHa. This is fairly unusual for a metal yet it is a known feature of copper and few others. Even at low temperature we will have reasonably converged calculation in both Zn and Cu with small k-point meshes.

Finally, notice that the converged energies (for the finest k-meshes) vary with the temperature. Energies of calculation done with different electronic temperature CAN NOT bet compared directly.

Sustainable electronic temperature And pseudo-potentials accuracy

Now, we must determine which electronic temperature are usable in our cases and if our pseudo-potentials are accurate enough. This last part can be performed first. We will do a structural relaxation of the Zn and Cu cells and check simply that the obtained lattice parameters agree with experimental values. For a real study we should obviously perform much more checking.

Cell relaxation is a matter of adding the following variables in your input file (your responsibility is to tune the associated values):

```
ionmov  2
optcell 2
strfact 100
ecutsm  0.5
tolmxf  1e-5
ntime   20
```

Effects of this addition is the following: `ionmov` will allow ions to relax — this is useful only for complex structure; `optcell` tells Abinit to optimize the cell geometry and size; `strfact` inform Abinit of the arbitrary factor imposed for

	Cu $11e^-$		Zn $12e^-$	
T=0.001 Ha	$a_0 = 6.693$ bohrs	2.0%	$a_0 = 4.818$ bohrs $c_0 = 9.408$ bohrs	4.3% 0.64%
T=0.01 Ha	$a_0 = 6.692$ bohrs	2.0%	$a_0 = 4.822$ bohrs $c_0 = 9.375$ bohrs	4.3% 0.29%
T=0.02 Ha	$a_0 = 6.691$ bohrs	2.1%	$a_0 = 4.832$ bohrs $c_0 = 9.302$ bohrs	4.1% 0.49%
T=0.03 Ha	$a_0 = 6.686$ bohrs	2.1%	$a_0 = 4.844$ bohrs $c_0 = 9.225$ bohrs	3.8% 1.3%
T=0.04 Ha	$a_0 = 6.633$ bohrs	2.9%	$a_0 = 4.866$ bohrs $c_0 = 9.102$ bohrs	3.4% 2.6%
T=0.05 Ha	$a_0 = 6.601$ bohrs	3.4%	$a_0 = 4.891$ bohrs $c_0 = 8.973$ bohrs	2.9% 4.0%

Table 2.1: Accuracy of lattice parameter for Cu and Zn computed with various smearing parameter.

comparisons of forces and stresses; `ecutsm` smoothies effect of cell changes; `tolmxf` is the stopping criterion for any kind of cell optimization; and finally `ntime` is the maximum allowed number of optimization steps.

The single (resp. two) electron potential for Cu (resp. Zn) gives lattice parameter $a_0 = 2.52$ Å (resp. $a_0 = 2.44$ Å $c_0 = 3.98$ Å) that is errors of about 35% (resp. 9% and 22%). Clearly those pseudopotentials can't be used to describe copper or zinc; even roughly.

As shown in table 2.1 the complete pseudo-potentials gives much better results with a max discrepancy of less than 5%. Accuracy of the the copper pseudopotential is reasonable. That of the zinc is poor. For a real case study we would have to try others based on different exchange correlation functionals

Effects of electronic temperature differs in our metals. While effects are very small for copper until 0.04 hartree (that is almost 12000 kelvins) the c parameter of zinc changes fast. Overall it appears very reasonable to use electronic temperature of at least 0.02 hartree. The resulting error is much smaller than the one resulting from the pseudo-potential/exchange correlation function.

2.3.3 Brass

We have now chosen the pseudo-potentials. The cutoff energy to choose is known. We need again to check the sampling of reciprocal space and equilibrium volume before getting into actual calculations of elastic properties. Again we would need to check the convergency of electronic temperature and kpoint length for this problem. This is done exactly as before.

Brillouin's zone sampling in brass

We again have to pay attention to convergence of the k-point mesh and potential effect of electronic temperature. For a simple cubic cell as that of brass it is very tempting to use directly the CCP mesh defined by:

```
nshiftk 4
shiftk 0.5 0.5 0.5
       0.0 0.5 0.5
       0.5 0.0 0.5
       0.5 0.5 0.0
```

However if we had to face a more difficult situation we could have relied on Abinit ability to self decide the convenient mesh by using the input variables `prtkpt` and `kptrlen` in a preparation run.

2.3.4 Elastic constants

It is possible to determine elastic constants by response function calculations. This is an advanced feature of Abinit and it looks like too much advanced for this lecture. However you are encouraged to have a look at Abinit's tutorial on elastic constant calculations. Here we will concentrate on getting them by finite difference method.

With the help of table 2.2 we can impose some strains to the cubic cell of brass 50%. Strains formulated in this table are derived to ease our computation by keeping the volume constant. Nonetheless we shall not choose the deformation parameter γ too bluntly.

First we must keep within the elastic limit: remember that even though the cell will not be allowed to relax we must relax atomic positions to get the correct ground state energy under the imposed strain. High values of γ may causes instabilities. Moreover the greater the deformation, the greater higher order elastic constants will enter the game and produce non-quadratic effects. To relax ions we must introduce the following keywords:

parameters	$\Delta E/V(\gamma)$
$\varepsilon_1 = -\varepsilon_2 = \gamma$ $\varepsilon_3 = (1-\gamma^2)^{-1} - 1$	$2C'\gamma^2 = (C_{11} - C_{12})\gamma^2$
$\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = \gamma$	$\frac{9}{2}B\gamma^2 = \frac{3}{2}(C_{11} + 2C_{12})\gamma^2$
$\varepsilon_6 = \gamma$ $\varepsilon_3 = \gamma^2(4-\gamma^2)^{-1}$	$\frac{1}{2}C_{44}\gamma^2$

Table 2.2: Elements of strain tensor (ε_n) and associated changes of energy for an elastic media. The strain tensor elements and elastic constants are expressed with Voigt's index contraction. This formulation is valid only for cubic materials.

ionmov: when set to 2 or 3 this variable will lead Abinit to relax atomic positions. For very small system 3 is the recommended value.

ntime: it is the parameter that commands the maximum number of relaxation steps. This can be set to any big integer value yet 10 to 20 are probably more than enough.

dilatmx and optcell: remember that we don't want to relax our cell. The default null values are OK for those two parameters.

tolmxf: The only efficient parameter to stop an atomic relaxation is vanishing forces. `Tolmxf` control the tolerance on the max forces. A value of 10^{-5} hartree per bohr is enough for our purposes. Note that the SCF convergence criterion shall be chosen accordingly: you must make sure that forces are converged beyond this stopping criterion.

Second, we may encounter problems arising from our discretization of space. Since we change cell parameters and axis lengths, the number of grid points included in our calculation may change. Since we use a heavyside function to discriminate between included plane-waves and excluded one, changes will result in numerical noise. To avoid this we can use the parameter `ecutsm`. This allow for a continuous, with continuous second derivative, cutoff functions. Usually `ecutsm` is chosen to be 0.5 hartree.

We have many ways to derives elastic constant. The safest is probably to evaluate for multiple values of γ the total energy. And then assess elastic constants. Note that Abinit is computing strain for each cell. You may check your result by deriving elastic constant from the stress-strain relation.

2.4 Project

Abinitio calculation is a versatile tool once mastered. Yet many errors can be done and it is quite important to master the basis. That's why hereafter we propose you a very simple project that consist in simply following step by step the calculations explained in this documents. We propose you to discover the mechanical properties of β-SiC. From the bare knowledge of it's crystal structure: zincblende. That is a face centered cubic where each atomic species has the other for nearest neighbors.

Let's remind the steps to get the elastic constant:

1. Get the correct cutoff energy for the provided pseudo-potentials of Si and C. Silicon form a cubic diamond structure with a lattice parameter of about 5.43 Å and carbon an hexagonal close packed whose lattice parameter are about $a = 2.46$ Å and $c = 6.71$ Å.

2. Silicon carbide is an insulator. We do not need an electronic temperature convergence study. We can directly do the k-point mesh convergence. Find an appropriate mesh for $\beta - SiC$ using `prtkpt`. Note that you have to guess the cell parameter.

3. Relax the cell parameter using `optcell`, `ecutsm`, `dilatmx`, `tolmxf`, and `strfact`..

4. If the relaxed cell parameter has greatly changed from you initial guess, you need to restart the k-point convergency and eventually relax again the cell.

5. Apply the formulas of table 2.2 to get the elastic constants..

Has an extra, you can also compute the properties of α-SiC: a wurtzite. That is is a closed packed hexagonal structure with two atomic species where each species have only the other as nearest neighbor. Beware that formulas of table 2.2 do not hold. You need to derive an other set of formulas to get the five independent, non-zero, constants for hexagonal materials: C_{11}, C_{12}, C_{13}, C_{33} and C_{55}.

Chapter 3

Abinit's features

The possibilities of Abinit are much wider than what we have learned in the previous part. However it is clearly beyond the scope of this summer school to detail all its capabilities. It is nonetheless interesting to sum them up just to let you know that, one they are available, two some tutorials detailing there uses are accessible at http://abinit/Infos_v5.2/tutorial/welcome.html.

3.1 Plane waves calculations

3.1.1 Parallel calculations

There are situations where a sequential code is not enough, often because it would take too much time to get a result. There are also cases where you just want things to go as fast as your computational resources allow it. To this end, the Abinit package provides a parallel version of the main binary, called abinip.

Abinip parallelism is based on MPI, the standard message passing interface (Note however that some elements of Abinit are parallelized with openMP yet this is not efficient actually). Since MPI is standardized many implementation exists. Among them two open-sources code are most frequently used together with Abinit: MPICH and openMPI.

The parallel version of Abinit is compiled and tuned by using the various mpi flags proposed by its build system (in Abinit directory type `configure --help $|$ grep mpi` for more details).

3.1.2 Spin calculations

Abinit like most DFT code can handle magnetism. This is frequently useful; not only for magnetic materials but also when trying to reproduce most accurately materials properties. You have a complete tutorial available on this matter.

3.1.3 GW

Although the Kohn and Sham bands are know to be usually reasonable, quantities like the band gap are, even qualitatively, wrong. An accurate method to get both correct band structure and band gap is the so called GW approximation. You can learn how to use it with Abinit's tutorials.

3.1.4 TDDFT

An other method to correct Kohn-Sham energies is the time dependent density functional theory (TDDFT). You can learn how to do such calculations in the Casida's approach with the tutorial on TDDFT.

3.1.5 PAW

We have witnessed in this lecture that the cutoff energy required to get converged results might sometimes be quite high with norm conserving pseudo-potentials like HGH. This is a really limiting factor for lots of calculations. That's why people have developed other kinds of pseudo-potential. One of the most effective (softer) kind is PAW. Using PAW requires to tune more parameter than the use of norm conserving pseudo-potential — more approximations $\Rightarrow$ more parameters — and you will probably be happy to find a tutorial on this technique which allow drastic reduction in computational time.

3.1.6 Various analysis tools

Analyzing and visualizing parts of Abinit output may be difficult, like all display of three dimensional functions. Abinit comes with a set of program among which one is used to convert Abinit output into readable file for visualization tools like "open DX". The tutorial on analysis tools will help you mastering the visualization problem.

3.1.7 Response function calculation

Calculations of quantities which are first, second and third derivative of total energy is a very large part of DFT uses. Thanks to the so called $2n + 1$ theorem all those can be derived from first order wave functions. There is many tutorials within Abinit to teach how to do those calculations. Going through them you will learn, for instance, how to compute phonons or elastic constants.

3.2 BigDFT

This part of Abinit is currently in development. Then the associated set of calculations is much reduced. Today BigDFT is able to output total energy and forces in non periodical situations. But tutorials are not ready.

The source of differences between plane wave codes and BigDFT is the use of wavelets as a basis for wave functions representation. Wavelets are localized in real space and in Fourier space and form a complete basis set. This makes them the best known candidate for accurate order N DFT calculations.

Chapter 4

Conclusion

This introduction is aimed at helping beginners with first Abinit use, as well of demonstration of standard DFT calculations. It is mostly a written reminder of the content of my talk. Please use it extensively to help you with your abinitio calculation.

Note that it is not intended as a masterpiece of English grammar or spelling. Yet any comment on this are welcome. Much more welcome are comment on contents.

After reading this, if you want to get deeper skill in abinitio calculations you are heartily invited in reading Abinit's documentation. Especially using input variables documentation and tutorials. Feel welcome to ask any question on Abinit's forum (forum@abinit.org). By following the community "netiquette" (www.abinit.org/community/?text=netiquette) you'll quite likely get answers.

If you fall in love with DFT, reminds that Abinit is licensed under GPL granting you the freedom to make your own research and uses of this code. Again, you are welcome to contribute.