ABINIT School 2019
January 21 – 25, 2019
Bruyères-le-Châtel, France

Tuning ABINIT (precision, convergence) – Hands-on session

By following this step-by-step tutorial, you should experiment how to change the default settings of ABINIT to make it converge faster.

As usual, create a working directory where you will copy the input files and execute the code. Let's call it `~tutorial_tuning_work`.
Copy the starting input file — called `ttuning/ttuning.in` — into the working directory. Create a subfolder `pseudo` in the working directory and copy the pseudopotentials from `ttuning/pseudo`.

Then, create a ABINIT "`files`" file in order to run ttuning.in using the pseudopotentials from the `pseudo` directory (1-Bismuth, 2-Iron, 3-Oxygen).

We can now start.

For this hands-on session, the working system is a 20-atom cell of *Perovskite* BiFeO3 in its (theoretical) cubic phase. Two oxygen vacancies were introduced in it and the cell was distorted;
The final number of atoms is 18.
Introducing defects in the cell makes it more difficult to converge with ABINIT.

First attempt: run ABINIT with this initial file.
To perform the calculations in an acceptable time, it is preferable to use about 200 processors.
Note that the automatic parallelization is activated: **autoparal** is on.

```
#MSUB -n 200
…
ccc_mprun abinit <files >log
```

Use the standard procedure to submit the calculation on the supercomputer (**ccc_msub**). You can use the attached `cobalt.sub` submission file.

The run stops after *20 iterations* without having really converged the electronic self-consistent cycle.

Now, you can play with various input parameters in order to evaluate their respective influence on the convergence rate.
Before modifying a parameter in the input file, don't hesitate to read ABINIT documentation (*Input Variables* section).

Parameters you can play with:

- The history size of the mixing scheme, **npulayit**.
- The parameters of the dielectric matrix used to precondition the density residual: **diemix**, **diemac**.
- The maximal number of iterations of the (iterative) diagonalization scheme, **nline**.
- The number of non-self-consistent iterations, i.e. the number of restarts of the iterative algorithm, **nnsclo**.
- The mixing scheme **iscf** (7=on the potential, 17=on the density).

What is the best compromise?
Do you succeed in making the code converge in less than 20 iterations?
In principle, with optimal settings, ABINIT should reach convergence for this system in less than 20 iterations...

Is it possible to further improve this result?
Let's try to play with the number of bands. Let's make an additional experiment:

- Increase slightly (10%) the **nband** parameter (setting `nband=110`) and try this new setting.

So far, we have not changed the parameters dedicated to parallelism.

We have let ABINIT adjust them automatically. Only MPI parallelism with distributed memory was used.

Let's try now to run the code using hybrid parallelism. For that, let's use multithreading and choose to distribute each MPI process over 14 tasks. This number corresponds to the number of CPUs on 1 socket of a Cobalt node (each node is made of 2 sockets).

You should now modify the submission script by inserting the following lines:

```
#MSUB -n 16
#MSUB -c 14
…
export OMP_NUM_THREADS=14
```

With these setting, you will use at most 16 MPI processes spread over 14 threads, i.e. at most 224 CPUs.

Then start the job.

What does ABINIT do?

It stops prematurely, finding no appropriate process/task distribution!

In fact, 14 is not a very practical number. However, in the `log` file, ABINIT give some advice. A list of more suitable **`nband`** values is given. `nband=112` is identified as the optimal value.

Change the value of `nband` in the input file (`nband=112`) and restart the code.

The code runs now and the iterations converge rather well. The distribution of processes – automatically determined – should be:

```
npkpt = 2, npband = 8, bandpp = 14
```

The LOBPCG diagonalization algorithm is used, with a block size equal to `npband x bandpp = 112`; this is the ideal block size (all bands in 1 block).

However, the computation time needed to perform the iterations is much longer than previously. This shows that the use of the keyword **`autoparal`** is not optimal.

What's the issue? How to do better?

Indication: the most effective parallelism is that using the k points; the present **`npkpt`** value is very small (`npkpt = 2`).

A solution (optimal set of parameters) is given in the ttuning-best.in file.

Let's try now to speed-up the calculation.

Decrease the value of the input parameter **`accuracy`**. Try to set **`accuracy=3`**, then **`accuracy=2`** and finally **`accuracy=1`**.
You should notice a speed-up of the calculation... and a change in the significant digits (look for instance at the value of the energy ETOT).

Is this loss of precision acceptable?
To evaluate this, you can try to perform a <u>structural relaxation</u> of the simulation cell. Add the following line in the input file and run ABINIT:

```
ionmov 22 ntime 20
```

Does the relaxation run in the same way with `accuracy=1` or `accuracy=4`?