ABINIT
The User's Manual

A Open Source ab initio *Electronic Structure Calculation Software*

The ABINIT Group

Website: http://www.abinit.org

22 May, 2004

This manual is for ABINIT version 4.3.

Welcome!

Dear user of ABINIT (in short: ABINITioner),

If this is the first time that you have access to ABINIT, or that you receive an ABINIT announcement, welcome!

On the Web site, you will find a lot of things, including installation notes for different versions of ABINIT, pseudopotentials, some utilities, benchmarks, and a list of FAQ.

If you follow carefully the installation notes on the Web site, for a particular version of the code, you will soon be able to read information files contained in the `~ABINIT/Infos` directory on your local machine.

We will use the name `~ABINIT` to refer to the directory that contains locally the ABINIT package, uncompressed. In practice, a version number is appended to this name, to give for example: ABINITv4.1.2. `~ABINIT` contains different subdirectories. For example, the present file, as well as other descriptive files, should be found in `~ABINIT/Infos`. Some of these files are written in html, and are accessible also from the web site: e.g. the tutorial and the list of all ABINIT input variables. The best entry point, after the installation, is likely the "new user's guide" (see chapter 4. Contact us if you have still questions after having gone through the different files.

The logic of version releases is as follows:

The full name of a version has three digits (for example, 4.1.2). The first digit is the slowly varying one (in average, it is changed each other year). It indicates the major efforts and trends in that version. At the level of 1.x.y ABINIT, the major effort was placed on the "ground–state" properties (total energy, forces, geometry optimisation, molecular dynamics ...). With version 2.x.y, response–function features (phonons, dielectric response, effective charges, interatomic force constants ...) were included. The main additional characteristics of version 3.x.y were the distribution under the GNU General Public Licence, the set–up of the documentation and help to the user through the web site in html format, and the availability of GW capabilities. The version 4.x.y should put a lot of effort in the speed of ABINIT (e.g. PAW), and its parallelisation.

When additional features are present, the second digit in incremented. So versions 1.x started with 1.0 and finished at 1.9, with more and more features added. Version 2.0 was released mid–July 1999, and version 2.1 mid–November 1999. Two versions differing by the last (third) digit have the same capabilities, but one with the largest last digit is more debugged than the other: version 2.0.4 is more debugged than 2.0.3 or 2.0.2, but no new features has been added (so likely, no additional bug!).

As mentioned above, version 3 of ABINIT (as well as the present version 4 and future versions) is distributed under the GNU General Public Licence. The GNU General Public License is intended to guarantee your freedom to share and change free software — to make sure the software is free for all its users. To protect these rights, one needs to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

Usually, a version of ABINIT is released only when it has been installed 'successfully' on 5 different platforms. A successful installation includes a battery of more than 300 tests, whose results are automatically analyzed (this does not garantee that the code is bug free, however, and the contribution of testers is crucial).

For those who plan to install ABINIT on a new platform:

- of course, try first to install it on one of the platforms on which it is usually installed, and get used to it;
- do not hesitate to contact us;
- when you have done the work, please report to us, in order for your work to be made available to others;
- for vector machines, note that the default FFT routines are optimized for superscalar architecture, and that you likely have to change the fourwf.f and fourdp.f routines.

Also the default non–local routines (*nloalg* = 3 or 4) are optimized for superscalar architecture, You should likely use *nloalg* = 1 or 2, theoretically optimized for vector machines.

For those who plan to develop new features: you will find the information for developpers in the subdirectory `~ABINIT/Infos/Notes_for_coding`, when the code is installed on your machine.

A few information not contained in `~ABINIT/Infos` are worth to mention here:

- the accuracy of the code has been checked against the Fritz–Haber Institute FHI98MD code, that also uses plane wave and pseudopotentials. For different atoms, in LDA or GGA (PBE), the total energies found by both codes for the same pseudopotentials differed by less than 0.0001 Ha. This should be largely enough for 99.9% of the applications. Other code comparisons have been done, with results even better sometimes. Still, only a small fraction of all capabilities have been checked carefully against other codes ...
- more than 200 atoms have been treated in runs on a Intel/Linux machine by A. Horsfield, including geometry optimisation. Still, the usual use of ABINIT is for cells of up to 30 atoms, for which it is relatively well optimized ...

Here are a few advices about the SCF convergence. For small unit cells, there should be no problem: use *iprcel* = 0 with *iscf* = 3 (likely the fastest) or *iscf* = 5 (more robust). For small molecules in a big box, use *iprcel* = 0 with *diemix* on the order of 0.3 ... 0.8, and *diemac* =1.0d0, again with *iscf* = 3 or *iscf* = 5. For a larger number of atoms (> 10), or long cells (more than 4 layers of a material), you might gain a lot by using *iprcel* = 45, but then you MUST include a few conduction bands (20%–30% of the valence bands) and play also with *diecut* (the default 2.2d0 might be a bit low). On some systems (metallic slabs + atom or molecule), it is possible to gain a factor of three ... or simply to be able to converge!

For those who would like to compare ABINIT with other codes, especially for speed, please note the following:

- there is no ultra–soft pseudopotentials in ABINIT;
- the non–local part of pseudopotential in ABINIT is presently not treated in real space;
- we believe the FFT and non–local pseudopotential subroutines are rather efficient (not yet the orthogonalisation).

Despite the FFT and non–local psp efficiency, ABINIT has likely little chance to outperform a code that uses ultra–soft pseudopotentials. However, ABINIT should have nice features that might not be found in some other codes, noticeably:

- response–functions are available, as well as
- GW computation of excitation energies, and,
- nice pre– and post–processing,
- convenient input format,
- interface to many pseudopotentials and pseudopotential codes,

- portability,
- robustness, and also ...
- you get access to the sources without any fee.

If you find a problem with ABINIT, please contact us, and try to follow the `~ABINIT/Infos/problem_report` prescriptions.

Thank you for your help in this project!

Xavier and the ABINIT group.

Contents

Chapter 1

Release Notes

Many thanks to the following contributors to the ABINIT project between September 2003 and March 2004:

J.-M. Beuken, M. Boulet, F. Bruneval, M. Coté, A. Garcia, D. Hamann, Hasegawa-san, Iguchi-san, J. Iniguez, J. Jansen, F. Jollet, M. Lee, M. Merli, Mikami-san, J.-P. Minet, Y.-M. Niquet, Nishida-san, A. Oganov, V. Olevano, V. Recoules, A. Roy, L. Sindic, I. Souza, M. Torrent, F. Tournus, D. Vanderbilt, M. Veithen, M. Verstraete, X. Wu

It is worth to read carefully all the modifications that are mentioned in the present file, and examine the links to help files or test cases ... This might take some time ... Please note the WARNINGS!

ABINIT Version 4.3, released on February 18, 2004.

Changes are written with respect to version 4.2.

1.1 WARNINGS

1. Several input variables, related to the GW calculations, have been renamed:
   - *nbndsto* is now *nbandkss*;
   - *ncomsto* is now *npwkss*;
   - the routine outstat is now outkss;
   - *ngwpt* is now *nkptgw*;
   - *ecutmat* is now *ecutsigx*;
   - *npwmat* is now *npwsigx*;
   - *nshmat* is now *nshsigx*;
   - *plasfrq* is now *ppmfrq*.
2. The file format "_EM1", used in GW, has been transformed to "_SCR", to avoid some undesirable effect in the multi–dataset mode (the last character of "_EM1" is a digit). Thus the variable *geteps* has been renamed *getscr*, and the input variable *irdeps* has been renamed *irdscr*.

1.2 Most noticeable achievements

1. The development of the features related to strain perturbation continues:
   - the computation of the piezoelectricity tensor has been implemented (D. Hamann), See `Test_v4` #65–66.

- the analysis of the DDB containing response to strain perturbation has been implemented: one start from clamped ion elastic tensor, clamped ion piezoelectric tensor, as well as clamped ion internal strains, in order to compute corresponding "relaxed ions" quantities (Xifan Wu and Don Hamann). See `Test_v4` #67-70.
- related documentation, from D. Vanderbilt: `~ABINIT/Infos/Theory/lr.pdf`.

So, in short: you can now compute the physical tensors (elastic constant and piezoelectricity coefficients).

Remember: there are still limitations in the use of this strain perturbation: LDA only, non–spin–polarized systems, no spin–orbit coupling.

2. The tutorial GW is now fully operational.

   See `~/Infos/Tutorial/welcome.html`. Contributions by V. Olevano, F. Bruneval and XG.

   There is also a considerable speed–up of sigma (*optdriver* = 4) on vector architecture (VOlevano) and on superscalar architectures (XG).

3. The help file for cut3d has been translated in html, and is now accessible from the Web.

4. One can use the MPI I/O library to access the files in parallel, on massively parallel machines (not clusters, where the temporaries are local to each machine), with a centralized disk space, to which all processors have access: in case of *mkmem* = 0, avoid degradation of performance (by MBoulet). See the input variable *accesswff*. One needs also to specify the `-DMPIO` compilation flag, see `~/Infos/makefile_macros_help`.

5. New Time–dependent XC kernels for the ACFD computation of the correlation energy have been implemented: the BPG kernel, the energy–optimized kernels for Dobson and Wang. That part of ABINIT has been cleaned completely. There are 7 new tests: `Test_v4` #80–86. Work by YM Niquet.

6. The finite–electric field formalism has been implemented (for insulators only, with *nsppol* = 1, *nspinor* = 1). Documentation is provided for the input variable *berryopt* = 4. Also consult the automatic test `Test_v4` #78. This work has been done by MVeithen, starting from the implementation done using an earlier version of ABINIT, by J. Iniguez and I. Souza.

7. The constrained–polarization formalism (Na Sai et al, PRB 66, 104108 (2002)) has been implemented in ANADDB, by MVeithen. There are 3 tests of this implementation, see `Test_v4` #75, 76, 77.

8. The berry phase routine has been rewritten, so that it works now in parallel. Please, see the variable *berryopt*. However, not all options available with the old berryphase routine are available with the new one. This work has been done by M Veithen.

1.3 Most noticeable changes for the developers

1. Due to modification 4 in section 1.2, there have been large modifications of all routines in which there were IOs to a wavefunction file. A library of routines to access the Wavefunction files has been written, and is located in `Src_0basis` (a directory in which parallel routines can be stored, unlike `Src_4iowfdenpot`). A new datastructure "wffile_type" has been created, to contain all the information that allows to access a wavefunction file. This also prepare the use of NetCDF. Work by MBoulet and XG.

2. The NetCDF library v3.5.0 has been included in the ABINIT package. It can be compiled thanks to 'make netcdf' (modifications of the `makefile_macros` might be needed — see `makefile_macros.PGI_dummy`, as an example, as well as the explanation given in the fourth section of `makefile_macros_help`). The set–up use in a ABINIT program is exemplified in the 'make abinetcdf' and 'make testabinet'. Work by JP Minet.

3. The XMLf90 library v1.1 of A. Garcia has been included in the ABINIT distribution. However, it is not yet used.

4. All the definitions of structured datatypes have been gathered in one unique file `~ABINIT/Src_defs/defs_datatypes`. Indeed, the previously imagined system, in which supposedly the datastructures most specifically linked to one Src directory would be defined in the corresponding defs file, was not followed in practice. Moreover, one had observed a large increase of the number of defs files, sometimes one for only one datatype. Finally, it was difficult to see to which file belong each definition. On the other hand, there is apparently little problem in having only one big file for all datatype definitions:

   - everybody knows in which file are the definitions;
   - with the single instruction "use defs_datatypes", all the datatypes are accessible for the compilation of a file;
   - there is no increase in CPUtime or memory due to this concentration.

5. The treatment of the sequential and parallel versions have been made more similar, thanks to the more extensive use of the mpi_enreg datatype (see, in `~/Src_defs/defs_datatypes`, the new records of this datatype: paral_compil_kpt and paral_compil_fft), as well as the encapsulation of all operations summing data over groups of processors. Because of this, most of the sections that were selected by the `-DMPI` flag do not call anymore the MPI library (only small subroutine do it), and the corresponding routines have been taken out of the `Src_8seqpar` directory (the latter contained routines whose sequential and parallel version differed). Work by MBoulet and XG.

6. The directory `Src_1managempi` has been created. It contains some routines previously in `Src_8seqpar` or `Src_0basis`. See modification 5.

7. Due to modification 2 in section 1.1, there have been numerous changes in the names of routines: rdem1 to rdscr, wrem1 to wrscr, testepsm1 to testscr, rdepsm1 to rdppscr.

8. The extension of all current *.htm files has been changed to *.html. The hyperlinks to these files have been updated.

9. The automatic tests are now driven by the script "RunTests", present in the main directory. Configuration files "*.cnf" are present in the Test subdirectories. The writing of automatic test has been largely simplified by this new script. Work by L. Sindic.

10. makemake has been modified so that CPP is used for all library directories (while it was only used for fftnew until now).

1.4 Other changes

(or on–going developments, not yet finalized).

1. Improvements to the Infos directory content (FAQ, Miscellaneous) have been made by Mikami-san.

2. New test J for parallelism: use of disk for GS and RF

3. Parallelization of berry phase calculations. Should be tested. By M Veithen.

4. Output of cut3d for Open DX: given now in Angstrom. From M Cote

5. New `makefile_macros` for "stardust" HP machine, from AOganov New `makefile_macros` for an Itanium2 machine, from Mikami-san

6. PAW in progress. Numerous modifications, for speed. From MTorrent and FJollet. As announced in v4.1, the PAW part of ABINIT is available for BETA TESTING, although forces and stresses remain still to be implemented. Two pseudopotentials are available in `~ABINIT/Psps_for_tests`, while for other pseudos ... use the translator!

7. Bug fix: *iexit* = 1 from gstate, not being related to an abinit.exit file. From Don Hamann.

8. For OpenVMS, use of VMS version numbers in case a file already exist. From JJansen.

9. New input variables: *boxcutmin*, *optforces*

10. The Fermi level has been generalized to the case of insulators: the so–called "HOMO" (highest occupied molecular orbital) energy is now delivered, in case of an insulator (*occopt* ==0, 1 or 2).

11. The average exchange–correlation potential is now printed in the main output file. Because the averages of the local potential and Hartree potential are set to zero (they are actually undefined), this is also the average of the local part of the Hamiltonian.

12. Work on delocalised coordinates and on electron–phonon interaction has started. Some routines are included in v4.3, but debugging is going on (M. Verstraete).

13. FFT parallelism is going on. Work by ARoy and XG.

14. In spin–polarized case, cut3d can deal now directly with the total density, not only the spin up, spin down or magnetization densities.

15. There is now a manual for the conducti utility: `conducti_manual.tex` (from V. Recoules).

16. Bug fixes, small modifications, etc ... by F. Tournus, M. Lee, Nishida-san, Mikami-san, M. Merli, J.-M. Beuken, D. Hamann, Hasegawa-san, F. Bruneval, M. Verstraete, A. Oganov, M. Torrent, A. Roy, M. Veithen, Y. Pouillon, J. Junquera, V. Recoules

Chapter 2

Features

Related documentation

The reader might consult the latest version of the 'context' file for the description of the ABINIT project and its history. The latest version of the 'planning' file will give an idea of future developments. The different versions of the `release_notes` files will allow to see the actual development of the project since version 1.5, released in August 1998. Then, there are also the "New User Guide" and "ABINIS Help" files, for accurate descriptions of the code and its use.

2.1 Available physical properties

2.1.1 Computation of the total energy of an assembly of nuclei and electrons placed in a repeated cell.

1. The computation is done using plane waves and pseudopotentials.

2. The total energy computation is done according to the Density Functional Theory. Most of the important local approximations (LDA) are available, including the Perdew–Zunger one. Two different local spin density (LSD) are available, including the Perdew Wang 92, and one due to M. Teter. The Perdew–Burke–Ernzerhof, the revPBE, the RPBE and the HCTH GGAs (spin unpolarized as well as polarized) are also available.

3. Self–consistent calculations will generate the DFT ground–state, with associated energy and density. Afterwards, a non–self–consistent calculation might generate eigenenergies at a large number of k–points, for band structures. The electronic Density–Of–States can be computed either within the tetrahedron method, or using a smearing technique.

4. The program admits many different types of pseudopotentials. There are two complete sets of pseudopotentials available for the whole periodic table, one of the Troullier–Martins type, one of the Goedecker type (this one include the spin–orbit coupling). Four codes are available to generate new pseudopotentials when needed. Two of them are able to generate pseudopotentials with a core hole, in order to compute core–level shifts. Two of them are able to generate GGA pseudopotentials. No ultra–soft pseudopotential can be used. Pseudopotentials can be mixed, to generate "alchemical" pseudoatoms.

5. Metallic as well as insulating systems can be treated. Schemes for determination of the occupation number include the Fermi broadening, the Gaussian broadening, the Gaussian–Hermite broadening, as well as the modifications proposed by Marzari. Finite temperatures can also be treated using a smearing scheme (Verstraete scheme).

6. The cell may be orthogonal or non–orthogonal. Any kind of symmetries and corresponding sets of k–point can be input, and taken into account in the computation.

7. The electronic system may be computed in the spin–unpolarized or spin–polarized case, with the possibility to impose occupation numbers of majority and minority spins, and the spins of the starting configuration. A specific option for efficient treatment of anti–ferromagnetism (Shubnikov groups) is available. The treatment of non–collinear magnetism is available, but only for the total energy (no forces, stresses, reponse–functions ...). The total magnetic moment of the unit cell can be constrained.

8. The total energy, forces, stresses, and electronic structure can be provided with the spin–orbit coupling included.

9. A finite electric field can be imposed on insulators, using the proper Hamiltonian for periodic systems based on Berry's phase.

10. The decomposition of energy in its different component (local potential, XC, hartree ...) is provided.

11. Inner electronic eigenvalues can be computed thanks to the minimisation of the residual with respect to a target energy value.

12. The charge density of states close to the Fermi energy can be output, and provide a simple approach to STM image calculations.

2.1.2 Derivatives of the total energy and eigenenergies

1. Hellman–Feynman forces are computed from an analytical formula, and corresponds exactly to the limit of finite differences of energy for infinitesimally small atomic displacements when the ground–state calculation is at convergence. This feature is available for all the cases where the total energy can be computed (except non–collinear magnetism).

   A correction for non–converged cases allows to get accurate forces with less converged wavefunctions than without it. The decomposition of the forces in their different components can be provided.

2. Stress can also be computed. This feature is available for all the cases where the total energy can be computed, except non–collinear magnetism. The decomposition of the stresses in their different components can be provided. A smearing scheme applied to the kinetic energy allows to get smooth energy curves as a function of lattice parameters and angles. Alternatively, a facility for correcting the computed stress by the Pulay stress is provided.

3. The polarization can be computed within the Berry phase formulation. This feature is available for insulators, magnetic or non–magnetic, but not yet when spin–orbit splitting is present. The localisation tensor can also be computed. Constrained–polarization calculations can be performed, following Na Sai et al, PRB 66, 104108 (2002).

4. Accurate responses to atomic displacements and homogeneous electric fields are available, and allows to compute the interatomic force constants, the Born effective charges, the dielectric constant, the phonon band structure. Symmetry characters of the phonons at Gamma are computed. Thermodynamical properties, like the free energy, the heat capacity or the entropy, can also be computed, in the quasi–harmonic approximation. Available for the GGAs (except the HCTH one), while for spin–orbit, only responses to atomic displacements are available.

5. Accurate responses to strain perturbations are available, and allow to compute, presently, the elastic constants ("rigid–ion" as well as "relaxed"), the piezoelectricity tensor, as well as the so–called "internal strains", change of forces to atoms due to a strain, or change of the stress due to an atomic displacement.

6. Approximate or accurate susceptibility matrix and dielectric matrix can be computed, thanks to a sum over states.

7. Derivatives of the electronic eigenenergies with respect to the wavevector can be computed analytically.

8. Optical conductivity can be computed, thanks to the Kubo–Greenwood formula.

9. The band–by–band decomposition of the Born effective charges, and the localisation tensor is available.

10. Selected bon–linear response coefficients can be computed thanks to the $2n + 1$ theorem of perturbation theory. At present:

    - the electro–optic coefficients
    - the Raman cross sections

2.1.3 Excited states

1. Computation of ionisation energies ($N \rightarrow N - 1$ electrons) and affinities ($N \rightarrow N + 1$ electrons) in the GW approximation.

2. Excited states of atoms and molecules (spin–singlet as well as spin–triplet) can be computed within TDDFT. Oscillator strengths are available.

2.1.4 Displacement of atoms, and changes of cell parameters

1. Different algorithms (Broyden; modified Broyden; Verlet with sudden stop of atoms) allows to find the equilibrium configuration of the nuclei, for which the forces vanish. The cell parameters can also be optimized concurently with the atomic positions. Specified lattice parameters, or angles, or atomic positions, can be kept fixed if needed.

2. Two molecular dynamics algorithm (Numerov or Verlet) allow to perform simulations in real (simulated) time. The displacement of atoms may be computed according to Newton's law, or by adding a friction force to it. Nose–Hoover thermostat is available with Verlet algorithm. Langevin dynamics is also available.

3. The code can provide an automatic analysis of bond lengths and angles, and the atomic coordinates in a format suitable for vizualisation with XMOL.

2.1.5 Analysis and graphical tools

1. A post–processor, called "cut3d", is available to analyse density, potential and wavefunction files. It is able to change the format of these files, to extract the data on a 2D plane, or along a 1D line. It can perform the Hirshfeld computation of atomic charges. It can analyse the charge contained in an atomic sphere, and determine the angular momentum projected charge (l=0 to 4) contained in that sphere.

2. Another post–processor, called "aim", is available to perform the Bader "Atom–In–Molecule" analysis of the density.

3. A special part in the tutorial (see later) indicates how to generate properly formatted data for the visualisation of:

    - the band structure (visualisation thanks to XMGR)
    - total energies vs different parameters (also using XMGR)

- the charge density (3D isosurfaces) (the cut3d postprocessor must be used, followed by matlab)

The cut3d postprocessor also allows to prepare 2D charge density plots.

4. The post–processor band2eps allows to draw phonon dispersion curves automatically, in a file written in Encapsulated PostScript (eps). Moreover, a color code allows to emphasize the contribution of individual atoms to the corresponding eigenvector (at most three types of atoms).

2.2 Speed and memory

2.2.1 Speed in the sequential version

1. Depending on the number of atoms, there are two regimes in the code: at low number of atoms and electrons, the CPU time is dominated by Fast Fourier Transforms with an average scaling $O(N^2 \log N)$ where N is some number characteristics of the size of the system (atoms, electrons); at large number of atoms and electrons, the CPU time is dominated by non–local operator aplication and orthogonalisation, with an average scaling $O(N^3)$.

2. The complex–to–complex Fast Fourier Transform routine for application of the Hamiltonian has been highly optimized, and take into account the fact that the wavefunction do not fill the reciprocal space FFT box. Library FFTs are also available, but they are found to be slower than the present FFT routine, developed starting from a routine provided by S. Goedecker.

   A real–to–complex FFT is used for treating potential and densities of the ground state, since they are real. For selected k–points, invariant under time–reversal symmetry, ((0 0 0), (1/2 0 0), (0 1/2 0), (0 0 1/2), (1/2 1/2 0) ... (1/2 1/2 1/2)), the number of planewave explicitly treated is divided by two. A real–to–complex FFT is used then.

3. The non–local potential is applied in reciprocal space. It has been optimized carefully, although there is still some speed–up to be coded when the k–point is invariant under time–reversal. The orthogonalisation procedure can be done twice per loop or only once.

4. At the level of the generation of electronic eigenfunctions, an efficient band–by–band preconditioned conjugate–gradient algorithm is used, in its non–self–consistent version.

5. At the level of the self–consistency loop, an efficient potential–based preconditioned conjugate–gradient algorithm is used. Simple mixing is also available, as well as the Anderson algorithm. Preconditioning of this algorithm is achieved through a model dielectric function, or through an approximate dielectric matrix.

2.2.2 Speed in the parallel version

1. For ground–state calculations, the code has been parallelized on the k–points, on the spins, on the bands, and on the FFT grid and plane wave coefficients.

   For the k–point and spin parallelisations (using MPI), the communication load is generally very small. This allows it to be used on a cluster of workstations.

   However, the number of nodes that can be used in parallel might be small, and depends strongly on the physics of the problem. The band parallelisation (also using MPI) can be used concurently with the k–point and and spin parallelisation, but is less efficient.

   The FFT grid parallelisation (using OpenMP) works only for SMP machines, and is still to be optimized. Alternatively, a MPI version is under development.

2. For response calculations, the code has been parallelized on k–points, spins, and bands (MPI), as well as on the FFT grid and plane wave coefficients (OpenMP).

   For the k–points, spins and bands parallelisation, the communication load is rather small also, and, unlike for the GS calculations, the number of nodes that can be used in parallel will be large, nearly independently of the physics of the problem.

   The FFT grid parallelisation (using OpenMP) works only for SMP machines, and is still to be optimized.

3. A careful study of the speed–up should still be done, in both the GS and RF cases.

2.2.3 Memory

1. The requirements of the different conjugate gradient algorithms on memory are relatively low, especially when the number of atoms is large. Optionally, it is even possible to use disk space to save memory, at the expense of computing time. In particular, when the number of k–points is large, they can be stored in memory one at a time. Phase factors in the application of the non–local operator can also be recomputed at each application, in order to save memory. For k–points that are invariant under time–reversal symmetry, the storage required for wavefunctions is half the storage for other k–point.

2.3 The user's point of view

2.3.1 The Web site

1. A Web site can be accessed. The complete sources (and all the tests) of the ABINIT package are available there. Executables for many different platforms are also available, in specific packages that also include the 'Infos' directory are also available. Installation notes, current features of ABINIT, the tutorial, on–line help can be vizualized directly from the web.

2. Also available from the Web site:

   - the pseudopotentials
   - some utilities (including cut3d, a density analyser),
   - four mailing lists (one for "official announcements", one for the developpers, one "forum mailing list" , one for the advisory commitee).
   - the ABINIT bibliography database, that contains references of papers in which ABINIT or one of its predecessors have been used.
   - the FAQ (frequently asked questions) database

2.3.2 Portability

1. The ABINIT package has been installed successfully on the following different platforms:

   - PC/Linux based on PPro, PII, PIII, PIV processors, with pghpf compiler, Intel compiler, fujitsu compiler, NAG compiler.
   - PC under Windows
   - ItaniumII
   - HP/SPP1600, HP/S–class, HP/N–class based on the HP 7200, 8000 and 8500 processors.
   - DEC alpha workstations under OSF, based on EV56, EV6, EV67 or EV68.
   - DEC alpha workstations under Linux, based on EV56.

- IBM RS6000 (models: 590, 3CT, nighthawk) based on Power 2 and 3+ processors.
- SGI Origin 2000
- CRAY T3E
- FUJITSU VPP–700
- Sun ultrasparc II
- NEC
- HITACHI SR8000
- Mac OS X

2. In particular, the parallel version is available on clusters of Intel/Linux, DEC or IBM workstations, as well as on CRAY T3E, SGI Origin 2000, HP/SPP1600, HP/S–class, HP/N–class, FUJITSU VPP–700 machines, HITACHI SR8000, IBM 44P, Sun ultrasparc.

3. Installation is made thanks to a sophisticated (but robust) suite of makefiles and scripts, and use a file preprocessor. Thanks to these, all machine–dependent parameters are grouped in one single short file for each machine. The parallel and sequential version of the code, as well as the different versions for the different machines, are prepared on–the–fly, by this suite of makefiles and scripts, so that there is only one unique source code.

4. Binaries for different machines can be managed in the same main directory, as they might be placed automatically in different sub–directories.

2.3.3 Running jobs, input and output files

1. The input variables are gathered in one unique file, read by a text processing facility build in the code. Many defaults values are provided, so that the input file can be kept rather short.

2. Many different stopping criteria allow the user to target the accuracy he or she wants to obtain.

3. The outputs are provided to one main file and one auxiliary file, as well as different specialized files (for density, potential, wavefunctions, ...). The main file is shorter than the auxiliary file, and well formatted, while all important results are gathered there. It can be used for archival purposes. The auxiliary (log) file will contain all exception messages.

4. Exception handling is provided through four different types of messages: COMMENT, WARNING, BUG and ERROR. In each case, the accurate meaning of the exception is described, as well as the eventual action to be taken by the user.

5. Statistics of foreseen memory and disk usage is printed at the beginning of the run. Statistics of CPU time usage is printed at the end of the run.

6. There is a facility to stop the run in a clean way at any time. The user may specify a cpu time limit, after which the job must end smoothly.

7. A status file, updated very frequently, gives an on–the–fly report of progress of the current run.

8. The code can handle multiple datasets contained in the input file, where generic input variables valid for all datasets can be defined. These calculations for different dataset can be chained, so that in one run, many complex tasks can be accomplished. This allows easy convergence studies.

9. The code can start the current run from a wavefunction input file generated in a previous run. This allows to cut down the number of iteration to self–consistency, or to perform other tasks. This can be done even if the previous job had different computational parameters, like different k–points, different energy cut–off, spin–unpolarized or spin–polarized wavefunctions, scalr or spinor wavefunctions ... Of course, the restart from wildly different parameters will not save a lot of CPU time.
10. ABINIT can produce CML (Chemical Markup Language) output files. It can also read the CML files it produced.

2.3.4 Documentation

1. A "New User Guide" and a few rather detailed help files (`abinis_help`, `respfn_help`, `anaddb_help`, `mrgddb_help`, `newsp_help`, `aim_help` ...) are available.
2. A tutorial is available. It starts with the computation of different properties of the H2 molecule, describes convergence studies, then focuses on bulk Silicon, bulk Al and Al surface. Finally, it describe the computation of dynamical and dielectric properties of AlAs.
3. Many test cases are provided, and can help the user in setting up a run.
4. An ABINIT bibliography database, that contains references of papers in which ABINIT or one of its predecessors is available on the Web site.

2.3.5 Generation of the k–points, geometries, and starting wavefunctions

1. The code can automatically generate symmetries from the primitive cell and the position of atoms. In this case, it identifies automatically the Bravais lattice, point group and space group. Alternatively, it can start from the symmetries and generate the atomic positions from the irreducible set. Also, a database of the 230 spatial groups of symmetry is built inside ABINIT.
2. Anti–ferromagnetic symmetry operations, and associated magnetic space groups (Shubnikov groups) can be treated with the same facility as usual space groups, including a database of the 1191 Shubnikov groups.
3. A geometry builder is available inside the code. It can take one (or two) group of atoms, rotate it, translate it and repeat it, then create vacancies.
4. The generation of special k–point sets (Monkhorst–Pack sets) and band structure k–points can also be done directly inside ABINIT. A list of interesting k–point sets, can be generated automatically, including a measure of their accuracy in term of integration within the Brillouin Zone.
5. A utility for generating wavefunctions with new characteristic (cut–off, k–point) from already existing wavefunctions with different characteristics is available (newsp).

2.3.6 Automatic determination of input parameters

1. Many defaults values are provided.
2. The FFT grid parameters can be automatically generated from the cut–off energy and geometry of the system.
3. The number of bands and starting occupation numbers can be automatically generated from the input set of atoms.
4. There is a database of atomic masses.

2.4 The programmer's point of view

1. The code is distributed without charge under the GNU General Public Licence (GPL). This garantees that the future modifications of the code stay available to the developpers and users for free. This Copyright is often referred to as a "Copyleft".

2. The code is written in clean Fortran90. Strict programming rules have been followed. These are documented. Comments are numerous, and all in english.

3. Quick, fully automatic, testing of the code is available in the makefile, giving diagnostics on the validity of computed energy, forces, stresses and eigenvalues for five typical cases.

4. More extensive testing is provided in five batteries of tests (altogether more than 350 different runs), with automatic comparison with results of preceding versions. A specialized diff script (called 'fldiff') has been written in order to ease the diagnostic on the suite of tests. In addition to tests of the correctness of the execution, for the sequential or parallel version of the main code, as well as some utilities, there are automatic diagnostics of the speed of crucial routines, and the response to a load of up to 4 instances of the main code, running concurrently.

5. Debugging facilities are provided inside the code, and can be directly accessed from the input file. The compilation can be done in both debugging or normal mode: the C–preprocessed files are either kept or removed automatically.

6. "Design–by–contract" utility routines are present, as well as a flag to preprocess the sources, and eliminate the design–by–contract features in production mode.

Chapter 3

Installation Notes

This file provides a description of the operations needed to install the ABINIT package, to generate the executable and to make the tests.

See a recent version of the new user's guide (chapter 4), for an introduction to the abinit package. See a recent version of the abinis help file (chapter 6) for learning how to use the code. Both of them can be found in the `Infos` subdirectory.

Any comment or suggestion to improve the procedure will be welcome! Simply contact the ABINIT group.

Created 1998/11/09.

Last updated XG 2004/02/18.

3.1 How to get a version of ABINIT?

We will distinguish three cases:

1. you have a F90 compiler under UNIX/Linux (or MacOS X) and you want to compile the source files.
2. you run one of the UNIX/Linux (or MacOS X) platforms on which ABINIT is installed before each release, and you do not want to compile the source files.
3. you want to run ABINIT under DOS/Windows.

In the three cases, the installation files are available on the web site.

You must download different files for the different cases(*x.x.x is the version*).

Case 1 (under UNIX/Linux or MacOS X, and you want to compile):

Download the file **src_tests_x.x.x.tar.gz** or download the four files **src_testsX_x.x.x.tar.gz** where **X** is **A**, **B**, **C** or **D**.

The **src_tests_x.x.x.tar.gz** gzipped tar file contains **all** the sources of the ABINIT code (including the files needed for generating the FFTs, NumRecip and Lapack libraries), the complete Infos directory, the complete set of Tests, all the scripts and makefiles, the pseudopotentials needed for tests.

It does NOT contain the object files and the binary executable files.

You might as well download separately four different tar.gz files named src_testsA_x.x.x.tar.gz, src_testsB_x.x.x.tar.gz src_testsC_x.x.x.tar.gz, and src_testsD_x.x.x.tar.gz. This opportunity is provided in case you have a slow connection to the ABINIT Web site, and think your ftp connection might be not stable enough to download the entire src_tests_x.x.x.tar.gz file: the four different files are each about one quarter of the single file, and altogether they contain also the full ABINIT package.

Case 2 (under UNIX/Linux or Mac OS X, and you do not want to compile):

Download **platform_seq_x.x.x.tar.gz**, for the sequential version only, or download **platform_par_x.x.x.tar.gz**, for the parallel version only (if it exists for the platform), or download **platform_x.x.x.tar.gz**, for sequential and parallel versions (if both exist for the platform).

Such gzipped tar files contains the binary executable files (sequential, or parallel, or both), the complete Infos directory, and the different files needed to execute either the 5 internal tests (sequential version), or tests from the `Test_paral` directory (parallel version), or both. In what follows, we will focus on the sequential executable (called abinis). One should begin to use the parallel executable (called abinip) only when sufficient experience has been gained with abinis.

The gzipped tar file does NOT contain the source files, neither the `Test_fast`, `Test_v1`, `Test_v2`, `Test_v3`, `Test_v4`, and `Test_cpu` directories, nor the corresponding pseudopotentials. The possible platforms are:

- pclinux_pgi (PC under Linux, using the Portland Group Inc. compiler),
- pclinux_ifc (PC under Linux, using the Intel compiler — a bit faster),
- ibm_pw2 (IBM with Power 2 processors),
- ibm_44p (IBM model 44P with Power III+ processors),
- compaq_ev56 (DEC/Compaq machines with EV56 alpha processors),
- compaq_ev67 (DEC/Compaq machines with EV67 alpha processors),
- hp_pa7200 (HP PA RISC processors, model 7200),
- hp_pa8000 (HP PA RISC processors, model 8000),
- hp_pa8500 (HP PA RISC processors, model 8500),
- sgi_r12000,
- sgi_r14000,
- sun_ultrasparc,
- fujitsu_vpp,
- nec,
- mac OS X.

For the Intel/Linux machines, the binaries contain all the library routines statically linked. They should be very portable (and they have been ported to more than 5 different machines in different countries, or of different construction). However, they have been compiled by a compiler that generates code that needs an IP number to work. So, they will not run if your machine has not been IP'ed. For the other platforms, the executable does not contain all the library routines (they are linked dynamically), so they might not be as portable. The portability of the executable has been checked only for IBM PW2 and SGI Origin 2000 platforms.

Case 3 (you want to run ABINIT under DOS/Windows):

Download **intel_DOSWin_seq_x.x.x.exe**, for the sequential version.

This is a self–extracting zip file that contains the binary executable files, the complete `Infos` directory, and the different files needed to execute the 5 internal tests. It does NOT contain the source files, nor the `Test_fast`, `Test_v1`, `Test_v2`, `Test_v3`, `Test_v4` and `Test_cpu` directories, and the corresponding pseudopotentials.

Note: in order to compile ABINIT under DOS/Windows, you need the PGI workstation environment. The procedure to be followed differs from what will be explained below, and will not be described. Contact the ABINIT group if needed.

In the three cases, execute the following actions:

1. Transfer the above–mentioned file(s) to your machine, in a directory referred to here as `~local_ABINITvx.x.x`. You should have about 100 MB of disk space to install the code, maybe more, depending on the version, and the number of tests that you will do.

2a Under UNIX/Linux, gunzip (on some machine you need `gzip -d`) and untar the file `src_tests_x.x.x.tar.gz` or `platform_x.x.x.tar.gz` (or the sequential or parallel reduction of the latter file):

```
gunzip src_tests_x.x.x.tar.gz | tar -xvf -
```

or

```
gunzip platform_x.x.x.tar.gz | tar -xvf -
```

2b Under Windows, double–click the icon of the file `intel_DOSWin_seq_x.x.x.exe` to retrieve the files and directories. You can also run it as a command in a DOS window or use your favorite unzipping utility.

If correctly done, a whole set of subdirectories should have been created. One of them is called 'Infos'. It contains many important informations. In particular, you will find the description of the different subdirectories in the `~local_ABINITvx.x.x/Infos/dirs_and_files` file. This file also describes the content of the Infos directory, that is, all the information files.

***** It is strongly adviced to the installer to read (and print) NOW the `~local_ABINITvx.x.x/Infos/dirs_and_filesfile`. *****

3.2 How to make the executables?

If you were in case 2, then you already have the binary executables abinis, newsp, mrgddb and anaddb, so that you might skip the present section and go to the internal testing (see section 3.3). However, just for fun, if you are in UNIX or Linux, you can issue the make command, in the `~local_ABINITvx.x.x` directory:

```
make
```

This will print the list of the basic keywords for the make utility. This list of basic keywords is completed by keywords specific to developers, obtained by typing

```
make dev
```

We now suppose that you have a F90 compiler and you want to compile the source files (case 1). You must provide to the 'make' utility some information: the location of F90 and C compilers on your machine, the cpp utility, the blas library, the location of PERL ... For this, you must create in the `~local_ABINITvx.x.x` directory a file (or a symbolic link) named `makefile_macros`, that you will have to design starting from already existing example files. Examples of such files are contained in the subdirectories of the `~local_ABINITvx.x.x/Machine_dept_files` directory. Actually, if one of the existing `makefile_macros` file is suitable for you, you will have simply to issue an instruction like the following:

```
ln -s Machine_dept_files/DEC/makefile_macros.newton makefile_macros
```

(here, a symbolic link is created, so that `makefile_macros` refers to `Machine_dept_files/DEC/makefile_macros.newton`, a file suitable for the DEC machine named 'newton' at the PCPM). In order to generate a new `makefile_macros` file, a help file called `help.makefile_macros` can be found in the `~local_ABINITvx.x.x/Infos` directory. You can ask some help to the ABINIT group. When you have succeeded to create one such file, please send it to us, so that it can be maintained in the next versions of the code.

When the `makefile_macros` file is ready, you will have to issue the 'make' command, followed by some keyword.

To get the sequential version of all the executables, you must type:

```
make allseq
```

or

```
make allseq >& log.file
```

(allseq is an abbreviation for *all sequential* executables; `log.file` can be any filename, and is useful when the messages from make are too long).

Issuing this 'make allseq' command will trigger a whole set of actions (not easy to follow, as you will see).

The make utility will use the file `Makefile`, that calls a script called `makemake`, to produce another makefile, called `tmp_makefile`, that calls the `makearch` script from different subdirectories, that create themselves `tmp_makefile` files, that compile the fortran source (and eventually one C file), create libraries and link the different objects.

Moreover, usually, this works without problem!

Let's us suppose that you do not issue the powerful 'make allseq' command, but that you want to create one–by–one the libraries and executables.

Then, for the sequential version, you must create:

- the lapack library (make lapack)
- the numrecip library (make numrecip)
- the new FFT library (make fftnew)
- the perl scripts (make perl)
- and on some platforms, the blas library (make blas)

Alternatively, the libraries can be created at once by issuing 'make libs'.

Then, you can issue 'make abinis' to make the sequential version of abinit, and 'make newsp' to make newsp, the wavefunction translator. The latter is needed for different tests of abinis (see below). You can also issue 'make mrgddb' to make the code that merge the derivative databases, 'make anaddb' to make the code that analyze interatomic force constant, 'make aim' for the Bader analysis utility, 'make cut3d' for the analyzer of three–dimensional density, potential or wavefunction files ...

In order to get a transferable executable on Intel/Linux machines, it might be needed to generate a static BLAS library. For this purpose, issue 'make blas'. Be consistent with this choice, at the level of the `makefile_macros` file. This possibility is also used for the Fujitsu machines.

Informations needed to generate the parallel version of ABINIT can be found in the `~local_ABINITvx.x.x/Infos/paral_use` file.

Suppose that something is going wrong in one of the above–mentioned steps, then, you can issue one of the 'clean_xxx' keywords (those mentioned by typing 'make') to clean the directory where something wrong happened.

With a bit of luck, you will succeed to generate the executables. You are ready to perform the tests.

In some cases, one wish to keep the binaries for different platforms in the same `~local_ABINITvx.x.x` directory. It is possible to make the compilation execute in such a way that the

result is stored in a machine–dependent subdirectory. Prior to the compilation on one platform, issue:

```
make bindir
```

Thanks to this command, a `tmp_bindir` file will be created, containing the definition of a variable $(BINDIR). Usually, this variable will be `BinAbinit/machine_architecture`, where `machine_architecture` is the result of the execution of the 'arch' command on the platform. This definition will be included in the make mechanism, and all the binaries will be put in $(BINDIR). For the compilation on another platform, the 'make bindir' command must be performed again, before compilation, and the binaries generated by that other compilation will be put in the corresponding directory. It is also possible to change the `tmp_bindir` file, to define another location, not obtained by the 'arch' command. Note that, if any, the cleaning will be directed to the $(BINDIR) directory, which is the expected behaviour. However, presently, the tests (internal as well as v1, and others) CANNOT use the executables in the machine–dependent location: if the tests are to be done, one must first copy the proper executable in the `~local_ABINITvx.x.x` directory.

3.3 How to make the internal tests? (sequential version only)

The abinis code has five small internal tests, that can be issued automatically, and that check themselves whether the results that have been obtained are right or wrong. At most 3 MB of memory is required, and 1 MB of disk space. PERL must be installed on your machine for these tests to work. You can get PERL from http://www.perl.org. In the DOS/Windows case, perl.exe MUST be accessible through the DOS PATH.

You can begin with the test number 1. Simply issue the command:

```
make test1
```

It will run during a few seconds. It will first print

```
cd Test_in; Run 1
Built-in test 1 will be run through dotest script
dotest: Starting built-in test 1
../abinis < test1.files > test1.log
```

then, you will eventually (if you are on a slow machine) see different instances of the status file, like the following:

```
Status file, with repetition rate 49 , status number 99

Level abinit: call gstate
Level gstate: call brdmin
Level brdmin: call scfcv_ini
Level scfcv: call vtorho
istep = 2
Level vtorho: compute rhog
```

then, the important information is as follows:

```
Status file, reporting on test 1

OK for total energy
OK for nuclei positions
OK for forces
OK for stresses
```

This means that the internal test 1 ran successfully. If you do not get this message, then the executables were not properly generated, or there is a problem with the make and scripts that drive the internal test. In this case, after having tried to solve the problem by yourself, you should contact somebody in the ABINIT group.

Note: the script detects the end of the run by using the 'ps' command following by a 'grep' command. This can lead to strange effects if a file whose name contain 'abinis' is currently visualized, or if another job is running under a name that contains 'abinis'. So, close your files before running the tests, and also check that there are no running jobs whose name contains 'abinis'.

Supposing test1 was OK, then you have to issue the command 'make tests'.

The test 1 will be done once more, followed by the 4 other internal tests. Again, we hope that you will get the positive diagnostics for the other tests. Altogether, these tests are about 50 sec on a Intel/PIII 450 MHz machine where no other job is running.

For further information on these internal tests, see the file `~local_ABINITvx.x.x/Test_in/README`.

You might now read the new user's guide (chapter 4), in order to learn how to use the code. This is useful if you consider that the installation has been successful, or if you want to continue the tests.

3.4 How to make the other tests (case 1 only)?

(Case 1 only, since in case 2, only the minimal testing tools have been transferred)

Although it is possible to make the other tests without knowing really how to use the code (since all steps involved — the run and subsequent analysis — are done automatically), it is the right time to read the new user's guide (chapter 4).

After this reading, you should look at the subdirectory `Test_fast`, and then `Test_v1` (and perhaps `Test_cpu`), where tests of the sequential version of ABINIT (abinis) can be done automatically. For tests of the parallel version (abinip) see the directory `Test_paral`, as well as the `~local_ABINITvx.x.x/Infos/paral_use` file. For tests of the response function features of abinis, and for tests of mrgddb and anaddb, see the subdirectories `Test_v2`. `Test_v3` and `Test_v4` presents further tests of recently implemented features of ABINIT.

In order to execute these tests, you will need a larger disk space than for the simple installation of the code (the total additional disk space required is on the order of 100 MB). After having checked the results of the tests, the user can easily retain the output files in a compressed form, and get rid off the wavefunction files by issuing

```
make decrease_size_tests
```

in the `~local_ABINITvx.x.x` directory. Separate commands are available for each of the Test directories.

You might as well remove all working directories by issuing

```
make clean_tests
```

in the `~local_ABINITvx.x.x` directory. Separate commands are available for each of the Test directories. This command remove all the subdirectories that begin with '_', which is the case of the working subdirectories. If you want to keep some of them, remove their initial '_', then issue 'make clean_tests'. You can still decrease the size of such renamed directories by the 'make decrease_size_tests' command.

1) `Test_fast` (for the sequential version only)

This subdirectory contains a basic set of tests of the code, aimed at testing whether the code is coherent in time (successive versions), and exercising many parts of the code. However, they do not examine its accuracy on physical problems, mainly because the number of plane waves used

is too small, and some tests are not run to self–consistent convergence. 32 MB of memory should be enough for these tests (with no other application running, however).

Read the `~local_ABINITvx.x.x/Test_fast/README` file carefully (at least the beginning). To run the tests, simply issue, in the main directory:

```
(RunTests machine_name fast) >& log
```

where `machine_name` will usually be the name of your machine (any other character string is fine, however).

The script will create a directory whose name will be build from the machine name and today's date. All the results will be in that directory. The output files will be automatically compared, thanks to a 'diff' command, to a set of reference files, either from the (old) Plane_Wave code (the corresponding difference files are prefixed by 'diff.'), or from a recent run of the ABINIT code (the corresponding difference files are prefixed by 'difnew.').

In addition to 'diff', another comparing tool called 'fldiff' — for 'floating diff' — is also used. It treats in a more clever way the comparison of floating numbers between the output files and the reference files. As used presently in the 'RunTests' script, for each run, only one single file, called 'fldiff.report', will be produced. If for one test case, the two files differ by the number of lines, the 'fldiff.report' file will report that it cannot compare the two files. Usually this problem will be seen at the level of 'command signs' appearing sometimes in the first column of the output files, so a typical error message (announcing something went wrong) will be:

```
Case_1
22
The diff analysis cannot be pursued: the command sign differ.
```

By contrast, it will identify the floating numbers and ignore their differences if they are within some prescribed tolerance (fldiff is kind of an expert system in this respect: for example, it is able to ignore the differences in timings). If everything goes fine for a test, fldiff should identify only the differences in:

- the dates of execution (eventually);
- the version numbers (eventually);
- the platform description (eventually);
- the overall execution time (this is ALWAYS printed, even without differences).

So, a successful execution of one test case may be announced as follows in the `fldiff.report` file:

```
Case_1
2
< Version 4.3.2 of ABINIT
> Version 4.3.0 of ABINIT
5
< Starting date: Fri 27 Jan 2004.
> Starting date: Sat 31 Jan 2004.
202
< +Overall time at end (sec): cpu= 7.1 wall= 8.0
> +Overall time at end (sec): cpu= 7.3 wall= 8.0
```

The `fldiff.report` file will have one such section for each `Test_case` that was run. It begins with the number of the test case, then includes a few blocks of three lines: the number of the line where something is happening, followed by the content of the two lines.

If differences (besides those described above) are found, then there is a problem, or the double precision tolerance needs to be adjusted (as time goes, this should become better). More information on the fldiff script can be found in the `~local_ABINITvx.x.x/Utilities/fldiff` file. If

needed, you can send by email the `fldiff.report` to the ABINIT group for help understanding it and fixing possible bugs.

2) `Test_v1`

This directory contains tests built in the same spirit as those in the `Test_fast` directory, but that exercise features not present in the Plane_Wave code, like the treatment of metals, the GGA, the new pseudopotentials, the multi–dataset mode, the cell parameters optimization, and the spatial symmetry groups database.

These were developed during the development time of the version 1 of ABINIT. Of course, the automatic difference procedure only compares to recent runs of the ABINIT code. As for the 'fast' test cases, the `fldiff.report` file is also available. 64 MB of memory should be enough for these tests (with no other application running, however). Usually, these tests are run with the sequential version of the code only. It is possible to make them using the parallel version of the code, but this need to define the COMMAND_PAR macro in your `makefile_macros` file.

3) `Test_v2`

This directory contains tests built in the same spirit as those in the `Test_fast` directory, but that exercise features not present in the Plane_Wave code, nor in version 1 of the ABINIT package, mainly the response function features, and the use of the mrgddb and anaddb codes. Again, the automatic difference procedure only compares to recent runs of the ABINIT code. As for the 'fast' test cases, the fldiff.report file is also available. 64 MB of memory should be enough for these tests (with no other application running, however).

4) `Test_v3` **and** `Test_v4`

This directory contains tests built in the same spirit as those in the `Test_fast` directory, but that exercise features not present in the Plane_Wave code, nor in version 1 or 2 of the ABINIT package, noticeably the use of the GW code, the utilities Cut3d, AIM, ... , the PAW ... Again, the automatic difference procedure only compares to recent runs of the ABINIT code. As for the 'fast' test cases, the `fldiff.report` file is also available. 64 MB of memory should be enough for these tests (with no other application running, however).

5) `Test_cpu` **(for the sequential version only)**

This subdirectory contains the scripts, and input files needed for testing the cpu time, either on progressively finer real space grids, or on progressively bigger unit cells. Please read the README file of this directory. Also for this suite of tests, you have simply to issue

```
(RunTests machine_name cpu) >& log
```

Unlike in the previous case, many directories will be created (more than 10 in the present version). Their name begins with the test name (A1, A2, A3, A4, B1, B2, B3, B4, C3, D3), and is followed by the machine name and the date. Inside these directories, many runs are done. There is a 'report' file that summarizes the timing of the different runs, and there is a 'diff' file, that compares these timings with the reference (output files from a PII at 450 MHz, usually). The structure of these tests is more complex than that of the `Test_fast` and `Test_v1` directories. The tools used are the 'serie' scripts (serieA, serieB, serieC and serieD) as well as the 'getrep' script. For an explanation, contact the ABINIT group. For the largest tests (B and D series), up to 200 MB of central memory are required.

6) `Test_paral` **(needs both abinis and abinip)**

This directory contains tests built in the same spirit as those in the `Test_fast` directory, but that exercise the parallel version of the ABINIT code. The 'Run' script in that directory (not in the

main directory) considers one of the different input files, and for this file, it will perform first a sequential run (using abinis), then use the parallel code (abinip) with one processing node, then perform different parallel runs with an increasing number of processing nodes. As for the other series of test, the diff and the fldiff utilities are used automatically.

7) How to run all the sequential tests, or a subset of these suited for developers?

In the main directory, it is possible to issue

```
make tests_allseq >& log
```

in which case, all the tests in `Test_fast`, `Test_v1`, `Test_v2`, `Test_v3`, `Test_v4`, `Tutorial`, `Test_cpu` and the sequential tests of `Test_paral` will be issued. The choice of the name of the machine is not let to the user: it is "test".

Also in the main directory, if you issue

```
make tests_dev >& log
```

the tests in `Test_fast`, `Test_v1`, `Test_v2`, `Test_v3`, `Test_v4`, and the sequential tests of `Test_paral` will be issued. As in the preceeding case, the name of the machine if fixed ("test"). In addition, a summary file, containing all the fldiff report files is created, then tarred and gzipped. It is given the name `summary_tests.tar.gz`, and might be sent by mail for the merge of a new contribution.

3.5 Things that are NOT in the installation files

- **Pseudopotentials**:

  The installation files contain a few pseudopotentials, for testing purposes. But many other pseudopotential have been generated. There exist presently two complete sets of ready–to–use LDA pseudopotentials, for the whole periodic table: those generated by Doug Allan and Alex Khein for the plane_wave code, and those published by Hartwigsen, Goedecker and Hutter (Phys. Rev. B 58, 3641 (1998)). Also, the Fritz–Haber–Institute code, that is freely available at http://www.FHI-Berlin.MPG.DE/th/fhi98md/fhi98PP, can be used to generate LDA as well as GGA pseudopotentials, that can be read by the code after minor modifications, see the Web site.

- The Web site http://www.abinit.org contains many other things ... The benchmarks results, the mailing lists, ...

- The community Web site http://cst-www.nrl.navy.mil/~erwin/abinit is an additional source of information.

3.6 For advanced users: how to make the installation files?

Case 1 (source and tests): you want to produce the file `src_tests_x.x.x.tar.gz`

Here, for example, we suppose that you have modified the code (in order to implement a new feature). This modification works, so it is time to synchronize with the main code. If the modifications are rather large, it is worthwhile to send your complete version to the ABINIT group.

First, you should give a version number to your version. Simply modify the appropriate variable in the Makefile file of your `~local_ABINIT` directory. Then issue

```
make new_version_number
```

to touch the few routines that need this version number, then

```
make allseq
```

as for the installation.

When you think everything can be frozen, you must issue

```
make src_tests
```

That's all..

Case 2 (version for a specific platform): you want to contribute a specific binary executable e.g. you work on a SGI machine, and want to produce the file `sgi_r14000_x.x.x.tar.gz`

The requirements are: all executables (abinis, newsp and abinip) are working, and you have not erased one of the files needed for the tests, both sequential and parallel. Also, the version number is supposed correct (see above to modify it). In this case, simply issue:

```
make sgi_r14000
```

For the sequential version only, or the parallel version only, use

```
make sgi_r14000_seq
```

or

```
make sgi_r14000_par
```

For the other platforms, use the appropriate platform name, mentioned at the beginning of the present document, or obtained by typing

```
make dev
```

Chapter 4

New User Guide

Foreword

The ABINIT package is written by the ABINIT group. See the files `~ABINIT/Infos/context` and `~ABINIT/Infos/planning` for more details about the ABINIT group and the ABINIT project.

You will find the welcome message, and basic information about the Web site in the welcome address.

Before reading the present file, you should get the paper "Iterative minimization techniques for *ab initio* total–energy calculations: molecular dynamics and conjugate gradients" M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos, Rev. Mod. Phys. 64, 1045–1097 (1992), and read the introductory section.

After having gone through this beginner's introduction, you should follow the tutorial.

4.1 Introduction

ABINIT is a package whose main program allows finding the total energy, charge density and electronic structure of systems made of electrons and nuclei (molecules and periodic solids) within Density Functional Theory, using pseudopotentials and a planewave basis. ABINIT also includes options to optimize the geometry according to the DFT forces and stresses, or to perform molecular dynamics simulation using these forces, or to generate dynamical matrices, Born effective charges, and dielectric tensors. In addition to the main ABINIT code, different utility programs are provided.

We will use the name `~ABINIT` to refer to the directory that contains the ABINIT package. In practice, a version number is appended to this name, to give for example : `ABINITv1.0.1`. `~ABINIT` contains different subdirectories. For example, the present file, as well as other descriptive files, should be found in `~ABINIT/Infos`. Other subdirectories will be described later.

4.2 The sequential version of ABINIT: abinis

The main code exists in a sequential version, with the name `abinis` (ABINIT sequential), and in a parallel version, with the name `abinip` (ABINIT parallel). In the present new user's help file, we will suppose that the sequential version is used. After installation, it is present in the package as `~ABINIT/abinis`.

To run `abinis` you need four things:

- Access to the executable, `abinis`.
- An input file.
- A files file (list of file names in a file).

- A pseudopotential input file for each kind of element in the unit cell.

With these items a job can be run.
The full list of input variables, all of which are provided in the single input file, is given in the ABINIT input variables file.
The detailed description of input variables is given in:

- Basic variables, VARBAS
- Developpement variables, VARDEV
- Geometry builder + symmetry related variables, VARGEO
- Ground–state calculation variables, VARGS
- GW variables, VARGW
- Files handling variables, VARFIL
- Parallelisation variables, VARPAR
- Projector–Augmented Wave variables, VARPAW
- Response Function variables, VARRF
- Structure optimization variables, VARRLX

A set of examples aimed at guiding the beginner is available in the tutorial. Other test cases (more than 200 input files) can be found in the `~ABINIT/Test_fast`, `~ABINIT/Test_v1`, and `~ABINIT/Test_v2` directories.
Many different sorts of pseudopotentials can be used with ABINIT. Most of them can be found on the ABINIT web site. There is a set of Teter hardness–conserving potentials, a set of Troullier–Martins potentials, a few Goedecker–Teter–Hutter pseudopotentials, and Hartwigsen–Goedecker–Hutter potentials for the whole periodic table. A subset of existing pseudopotentials are used for test cases, and are located in the `~ABINIT/Psps_for_tests` directory. Information on pseudopotential files can be found in the ABINIT help file and the `~ABINIT/Infos/Psp_infos` directory.

4.3 Other programs in the ABINIT package

In addition to abinit, there are utility programs. `mrgddb`, `anaddb`, `aim`, `conducti`, `newsp`, and `cut3d` are present in the package. Others (presently `kptgen`) might be found on the Web site.
`mrgddb` and `anaddb` allow to post–process reponses to atomic displacements and/or to homogeneous electric field, as generated by abinit, to produce full phonon band structures, or thermodynamical functions. "mrgddb" is for "Merge of Derivative DataBases", while "anaddb" is for "Analysis of Derivative DataBases".
Another utility is `newsp`, whose main routine source is called `newsp.f`. It allows a crude interpolation among the wavefunctions at different k–points and is useful in reformatting wavefunction files to restart jobs on either new unit cell geometries, new planewave cutoffs, or new k–point grids. Most of its capabilities have been transferred recently inside abinit, however.
`cut3d` can be used to post–process the three–dimensional density (or potential) files generated by abinit. It allows to deduce charge density in selected planes (for isodensity plots), along selected lines, or at selected points. It allows also to make the Hirshfeld decomposition of the charge density in "atomic" contributions.
`aim` is also a post–processor of the three–dimensional density files generated by abinit. It performs the Bader Atom–In–Molecule decomposition of the charge density in "atomic" contributions.

`conducti` allows to compute the frequency–dependent optical conductivity.

A last one is `kptgen`, that allows to find the symmetries of a set of atoms in a unit cell, and to generate grids of k–points. It is available on the Web site. All of its capabilities are present inside abinit, however, and are even more sophisticated.

At the level of graphics, many commercial or free softwares can be used to visualize ABINIT outputs. Some indications are contained in the `~ABINIT/Infos/Tutorial/lesson_visual` file, but this topics has not yet been the subject of a systematic help file.

4.4 Input variables to abinit

The ABINIT help file describes the input variables and the output file. As an overview, the most important input variables are listed below:

acell(3)	lattice constant of periodic cell in bohr.
ecut	planewave kinetic energy cutoff in hartree.
ionmov	when *ionmov* = 0: the ions and cell shape are fixed = 2: search for the equilibrium geometry = 6: molecular dynamics
iscf	either a positive number for defining self–consistent algorithm (usual), or -2 for band structure in fixed potential
kptopt	option for specifying the k–point grid. If *kptopt* = 1, automatic generation, using *ngkpt* and *shiftk*. (for the latter, see abinis help)
natom	total number of atoms in unit cell
ngkpt(4)	dimensions of the three–dimensional grid of k–points
nstep	maximal number of self–consistent cycles (on the order of 20)
ntime	number of molecular dynamics or relaxation steps.
ntypat	number of types of atoms
occopt	set the occupation of electronic levels: = 1 for semiconductors = 3 . . . 7 for metals
rfelfd	when $\neq 0$: will do response calculation to electric field
rfphon	when = 1: will do response calculation to atomic displacements
rprim(3,3)	dimensionless primitive translations of periodic cell; each COLUMN of this array is one primitive translation
typat(*natom*)	sequence of integers, specifying the type of each atom. NOTE: the atomic coordinates (xangst, xcart or xred) must be specified in the same order
tolmxf	force tolerance for structural relaxation in Hartree/Bohr
tolvrs	tolerance on self–consistent convergence
xangst(3,*natom*)	cartesian coordinates (Angstrom) of atoms in unit cell. NOTE: only used when *xred* and *xcart* are absent
xcart(3,*natom*)	cartesian coordinates (Bohr) of atoms in unit cell. NOTE: only used when *xred* and *xangst* are absent
xred(3,*natom*)	fractional coordinates for atomic locations; NOTE: leave out if *xangst* or *xcart* is used
znucl(*ntypat*)	Nuclear charge of each type of element; must agree with nuclear charge found in psp file.

4.5 Output files

Output from a abinis run shows up in several files and in the standard output. Usually one runs the command with a pipe of standard output to a log file, which can be inspected for warnings or error messages if anything goes wrong or otherwise can be discarded at the end of a run. The

more easily readable formatted output goes to the output file whose name is given in the "files" file, i.e. you provide the name of the formatted output file. No error message is reported in the latter file. On the other hand, this is the file that is usually kept for archival purposes.

In addition, wavefunctions can be input (starting point) or output (result of the calculation), and possibly, charge density and/or electrostatic potential, if they have been asked for. These three sets of data are stored in unformatted files.

The Density Of States (DOS) can also be an output as a formatted (readable) file. An analysis of geometry can also be provided (GEO file) The name of these files is constructed from a "root" name, that must be different for input files and output files, and that is provided by the user, to which the code will append a descriptor, like WFK for wavefunctions, DEN for the density, POT for the potential, DOS for the density of states ...

There are also different temporary files. A "root" name should be provided by the user, from which the code generate a full name. Amongst these files, there is a "status" file, summarizing the current status of advancement of the code, in long jobs. ABINIT `abinis_help` contains more details.

4.6 What does the code do?

The simplest sort of job computes an electronic structure for a fixed set of atomic positions within a periodic unit cell. By electronic structure, we mean a set of eigenvalues and wavefunctions which achieve the lowest (DFT) energy possible for that basis set (that number of planewaves). The code takes the description of the unit cell and atomic positions and assembles a crystal potential from the input atomic pseudopotentials, then uses either an input wavefunction or simple gaussians to generate the initial charge density and screening potential, then uses a self–consistent algorithm to iteratively adjust the planewave coefficients until a sufficient convergence is reached in the energy.

Analytic derivatives of the energy with respect to atomic positions and unit cell primitive translations yield atomic forces and the stress tensor. The code can optionally adjust atomic positions to move the forces toward zero and adjust unit cell parameters to move toward zero stress. It can performs molecular dynamics. It can also be used to find responses to atomic displacements and homogeneous electric field, so that the full phonon band structure can be constructed ...

In order to know more about ABINIT, please follow the Tutorial.

Chapter 5

Tutorial

This tutorial is aimed at teaching the use of ABINIT, in the UNIX/Linux OS and its variants (OSF, HP–UX, AIX ...). It might be used for other operating systems, but the commands have to be adapted.

Note that it can be accessed from the ABINIT web site as well as from your local `~ABINIT/Infos/Tutorial/welcome.html` file. The latter solution is of course preferable, as the response time will be independent on the network traffic.

At present, six lessons are available. Each of them is at most two hours of student work. Lessons 1–4 cover basics, other lectures are more specialized.

Before following the tutorial, you should have read the "new user's guide", as well as the pages 1045–1058 of the paper "Iterative minimization techniques for *ab initio* total–energy calculations: molecular dynamics and conjugate gradients", by M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias and J. D. Joannopoulos, Rev. Mod. Phys. 64, 1045 (1992).

After the tutorial, you might find useful to learn about the tests cases contained in directories `~ABINIT/Test_fast`, `~ABINIT/Test_v1`, `~ABINIT/Test_v2`, `~ABINIT/Test_v3` and `~ABINIT/Test_v4`, that provide many example input files. You should have a look at the README files of these directories.

Additional informations can be found in the `~ABINIT/Infos` directory, including the description of the ABINIT project, guide lines for developers, more on the use of the code (tuning) ...

Brief contents:

Lessons 1–4 present the basic concepts, and form a whole: you should not skip one of these.

- Lesson 1 deals with the H2 molecule: get the total energy, the electronic energies, the charge density, the bond length, the atomization energy.
- Lesson 2 deals again with the H2 molecule: convergence studies, LDA versus GGA.
- Lesson 3 deals with crystalline silicon (an insulator): the definition of a k–point grid, the smearing of the cut–off energy, the computation of a band structure, and again, convergence studies ...
- Lesson 4 deals with crystalline aluminum (a metal), and its surface: occupation numbers, smearing the Fermi–Dirac distribution, the surface energy, and again, convergence studies ...

Lessons 5 and beyond present more specialized topics. You can pick one of these at random: with lessons 1–4 you know enough to start one of the others.

- The fifth lesson deals with the dynamical and dielectric properties of AlAs (an insulator): phonons at Gamma, dielectric constant, Born effective charges, LO–TO splitting, phonons in the whole Brillouin zone (in the future, it should also present the interatomic forces and the computation of thermodynamical properties).
- The sixth lesson deals with the computation of the quasi–particle band structure of Silicon, in the GW approximation (so, much better than the Kohn–Sham LDA band structure)

That's all for now ...
The following topics should be covered later:

- the choice of pseudopotentials.

5.1 Lesson 1: The H2 molecule, without convergence studies

This lesson aims at showing how to get the following physical properties:

- The (pseudo)total energy;
- The bond length;
- The charge density;
- The atomization energy.

You will learn about the two input files, the basic input variables, the existence of defaults, the actions of the preprocessor, and the use of the multi–dataset feature. You will also learn about the two output files as well as the density file.

This first lesson covers the sections 1, 3, 4 and 6 of the `abinis_help` file.

The very first step is a detailed tour of the input and output files: you are like a tourist, and you discover a town in a coach. You will have a bit more freedom after that first step ...

It is supposed that you have some good knowledge of UNIX/Linux.

This lesson should take about 2 hours to be done.

5.1.1 Computing the total energy, and some associated quantities

This is the first step (the most important, and the most difficult!).

Note that the present tutorial will use four different windows: one to visualize the html text of the tutorial (the present windows), a second to run the code, a third to visualize sections of the `abinis_help` file (that will open automatically), and a fourth one for the html input variables (that will also open automatically). Try to manage adequately these four windows ...

1. In addition to the present windows, open the second windows. Go to the Tutorial directory (that we refer as `~ABINIT/Tutorial`).

   ```
   cd ~ABINIT/Tutorial
   ```

   In that directory, you will find the necessary input files to run the examples related to this tutorial. Take a few seconds to read the names of the files and directories already present in `~ABINIT/Tutorial`.

2. You also need a working directory. So, you should create a subdirectory of this directory, whose name might be "Work" (so `~ABINIT/Tutorial/Work`). Change the working directory of windows 2 to "Work":

```
mkdir Work
cd Work
```

You will do most of the actions of this tutorial in this working directory. Copy the file `t1x.files` in "Work":

```
cp ../t1x.files .
```

3. Edit the `t1x.files`. It is not very long (only 6 lines). It gives the informations needed for the code to build other file names ... You will discover more about this file in the section 1.1 of the `abinis_help` file. Please, read it now (the third window shows up when you click on this link).

4. Modify the first and second lines of `t1x.files` file, so that they read:

```
t11.in
t11.out
```

Later, you will again modify these lines, to treat more cases. Close the `t1x.files` file. Then, copy the file `~ABINIT/Tutorial/t11.in` in "Work":

```
cp ../t11.in .
```

Also later, we will look at this file, and learn about its content. For now, you will try to run the code. Its location is `../../abinis`, So, in the Work directory, type:

```
../../abinis < t1x.files >& log
```

Wait a few seconds ... it's done! You can look at the content of the Work directory.

```
ls
```

Different output files have been created, including a "log" file and the output file "t11.out". To check that everything is correct, you can make a diff of `t11.out` with a reference file (that used slightly different names):

```
diff t11.out ../Refs/t11.out | more
```

You should get inoffensive differences, like differences in the name of input files or timing differences, like the following:

```
5c5
<  Starting date : Tue  4 Jul 2000.
---
>  Starting date : Thu 22 Jun 2000.
7c7
< - input  file    -> t11.in
---
> - input  file    -> ../t11.in
9,10c9,10
< - root for input  files -> t1xi
< - root for output files -> t1xo
---
> - root for input  files -> t11i
> - root for output files -> t11o
```

```
214c214
< - Total cpu        time (s,m,h):          4.7        0.08      0.001
---
> - Total cpu        time (s,m,h):          4.6        0.08      0.001
221,229c221,228
```

(... and what comes after that is related only to timing ...). If you do not run on a PC under Linux, you might also have small numerical differences, on the order of 1.0×10^{-10} at most.

If you get something else, you should ask for help!

Supposing everything went well, we will now detail the different steps that took place: how to run the code, what is in the "t11.in" input file, and, later, what is in the "t11.out" and "log" output files.

5. Running the code is described in the section 1.2 of the `abinis_help` file. Please, read it now.

6. It is now time to edit the `t11.in` file. You can have a first glance at it. It is not very long: about 40 lines, mostly comments. Do not try to understand everything immediately. After having gone through it, you should read general explanation about its content, and the format of such input files in the section 3.1 of the `abinis_help` file.

7. You might now examine in more details some input variables. An alphabetically ordered index of all variables is provided, and their description is found in the following files:

   - Basic variables, VARBAS;
   - Development variables, VARDEV;
   - Files handling variables, VARFIL;
   - Geometry builder + symmetry related variables, VARGEO;
   - Ground–state calculation variables, VARGS;
   - GW variables, VARGW;
   - Internal variables, VARINT;
   - Parallelization variables, VARPAR;
   - Response Function variables, VARRF;
   - Structure optimization variables, VARRLX.

   However, the number of such variables is rather large! Note that a dozen of input variables were needed to run the first test case. This is possible because there are defaults values for the other input variables. When it exists, the default value is mentioned at the end of the section related to each input variable, in the corresponding input variables file. Some input variables are also preprocessed, in order to derive convenient values for other input variables. Defaults are not existing or were avoided for the few input variables that you find in `t11.in`. These are particularly important input variables. So, take a few minutes to have a look at the input variables of `t11.in`:

   - *acell*,
   - *ntypat*,
   - *znucl*,
   - *natom*,
   - *typat*,
   - *xcart*,

- *ecut*,
- *nkpt*,
- *nstep*,
- *toldfe*,
- *diemac*,
- *diemix*.

Have also a look at *kpt* and *iscf*.

It is now time to have a look at the two output files of the run.

8. First, edit the "log" file. You can begin to read it. It is nasty. Jump to its end. You will find there the number of WARNINGS and COMMENTS that were issued by the code during execution. You might try to find them in the file (localize the keywords "WARNING" or "COMMENT" in this file). Some of them are for the experienced user. For the present time, we will ignore them. You can find more information about messages in the log file in the section 6.1 of the `abinis_help` file.

9. Then, edit the "t11.out" file. You find some general information about the output file in section 6.2 of the `abinis_help` file. You should also:

   - examine the header of "t11.out",
   - examine the report on memory needs (do not read each value of parameters),
   - examine the echo of preprocessed input data,

   until you reach the message:

   ```
   chkinp : Checking input parameters for consistency.
   ```

   If the code does not stop there, the input parameters are consistent. At this stage, many default values have been provided, and the preprocessing is finished.

   It is worth to come back to the echo of preprocessed input data. You should first examine the "t11.in" file in more details, and read the meaning of each of its variables in the corresponding input variables file, if it has not yet been done. Then, you should examine some variables that were NOT defined in the input file, but that appear in the echo written in "t11.out":

   - "nband": its value is 2.
     It is the number of electronic states that will be treated by the code. It has been computed by counting the number of valence electrons in the unit cell (summing the valence electrons brought by each pseudopotential) then occupying the lowest states (look at the "occ" variable), and adding some states (at least one, maybe more, depending on the size of the system).
   - "ngfft": its value is 30 30 30.
     It is the number of points of the three–dimensional FFT grid. It has been derived from "ecut" and the dimension of the cell ("acell").

     The maximal number of plane waves "**mpw**" is mentioned in the memory evaluation section: it is **752**.

     Well, this is not completely right, as the code took advantage of the time–reversal symmetry, valid for the k–point (0 0 0), to decrease the number of planewave by about a factor of two.

     The full set of plane waves is **1503** (see later in the "t11.out" file).

     The code indicates the time–reversal symmetry by a value of *istwfk*= 2, instead of the usual *istwfk*= 1 default.

- "nsym": its value is 16.
  It is the number of symmetries of the system. The 3×3 matrices *symrel* define the symmetries operation. In this case, none of the symmetries is accompanied by a translation, that would appear in the variable "tnons". The code did an automatic analysis of symmetries. They could alternatively be set by hand, or using the symmetry builder (to be described later).
- "xangst" and "xred" are alternative ways to "xcart" to specify the positions of atoms within the primitive cell.

Now, you can start reading the description of the remaining of the `t11.out` file, in the section 6.3 of the `abinis_help` file. Look at the `t11.out` file at the same time.

10. You have read completely an output file!

    Could you answer the following questions?

    - Q1. How many SCF cycles were needed to have the *toldfe* criterion satisfied?
    - Q2. Is the energy likely more converged than *toldfe*?
    - Q3. What is the value of the force on each atom, in Ha/Bohr?
    - Q4. What is the difference of eigenenergies between the two electronic states?
    - Q5. Where is located the maximum of the electronic density, and how much is it, in electrons/Bohr^3?

    (answers are given at the end of the present file)

5.1.2 Computation of the interatomic distance (method 1).

1. *Starting from now, everytime a new input variable is mentioned, you should read the corresponding descriptive section in the ABINIT help.*

   We will now complete the description of the meaning of each term: there are still a few indications that you should be aware of, even if you will not use them in the tutorial. These might appear in the description of some input variables ... For this, you should read the section 3.2 of the `abinis_help` file.

2. There are three methodologies to compute the optimal distance between the two Hydrogen atoms:

   - one could compute the **TOTAL ENERGY** for different values of the interatomic distance, make a fit through the different points, and determine the minimum of the fitting function;
   - one could compute the **FORCES** for different values of the interatomic distance, make a fit through the different values, and determine the zero of the fitting function;
   - one could use an automatic algorithm for minimizing the energy (or finding the zero of forces).

   We will begin with the computation of energy and forces for different values of the interatomic distance. This exercise will allow you to learn how to use multiple datasets.

   The interatomic distance in the `t11.in` file was 1.4 Bohr. Suppose you decide to examine the interatomic distances from 1.0 Bohr to 2.0 Bohr, by steps of 0.05 Bohr. That is, 21 calculations.

   If you are a UNIX guru, it will be easy for you to write a script that will drive these 21 calculations, changing automatically the variable "xcart" in the input file, and then gather all the data, in a convenient form to be plotted.

Well, are you a UNIX guru? If not, there is an easier path, all within ABINIT!

This is the multi–dataset mode. Detailed explanations about it can be found in sections 3.3, 3.4, 3.5 and 3.6, of the `abinis_help` file.

3. Now, can you write an input file that will do the computation described above (interatomic distances from 1.0 Bohr to 2.0 Bohr, by steps of 0.05 Bohr)? You might start from `t11.in`. Try to define a series, and to use the "getwfk" input variable (the latter will make the computation much faster).

   You should likely have a look at the section that describes the "irdwfk" and "getwfk" input variables: in particular, look at the meaning of **getwfk** -1.

   Also, define explicitly the number of states (or supercell "bands") to be one, using the input variables "nband". The input file `~ABINIT/Tutorial/t12.in` is an example of file that will do the job, while `~ABINIT/Tutorial/Refs/t12.out` is an example of output file. If you decide to use the `~ABINIT/Tutorial/t12.in` file, do not forget to change the file names in the `t1x.files` file ...

   So, you run the code with your input file (this might take one or two minutes), examine the output file quickly (there are many repetition of sections, for the different datasets), and get the output energies gathered in the final echo of variables:

```
etotal1  -1.0368223891E+00
etotal2  -1.0538645432E+00
etotal3  -1.0674504850E+00
etotal4  -1.0781904896E+00
etotal5  -1.0865814785E+00
etotal6  -1.0930286804E+00
etotal7  -1.0978628207E+00
etotal8  -1.1013539124E+00
etotal9  -1.1037224213E+00
etotal10 -1.1051483730E+00
etotal11 -1.1057788247E+00
etotal12 -1.1057340254E+00
etotal13 -1.1051125108E+00
etotal14 -1.1039953253E+00
etotal15 -1.1024495225E+00
etotal16 -1.1005310615E+00
etotal17 -1.0982871941E+00
etotal18 -1.0957584182E+00
etotal19 -1.0929800578E+00
etotal20 -1.0899835224E+00
etotal21 -1.0867972868E+00
```

   You might try to plot these data (see fig. 5.1. The minimum of energy in the above list is clearly between dataset 11 and 12, that is:

```
xcart11 -7.5000000000E-01  0.0000000000E+00  0.0000000000E+00
         7.5000000000E-01  0.0000000000E+00  0.0000000000E+00
xcart12 -7.7500000000E-01  0.0000000000E+00  0.0000000000E+00
         7.7500000000E-01  0.0000000000E+00  0.0000000000E+00
```

   corresponding to a distance of H atoms between 1.5 Bohr and 1.55 Bohr. The forces vanish also between 1.5 Bohr and 1.55 Bohr:

```
fcart11 -5.4963645520E-03  0.0000000000E+00  0.0000000000E+00
```

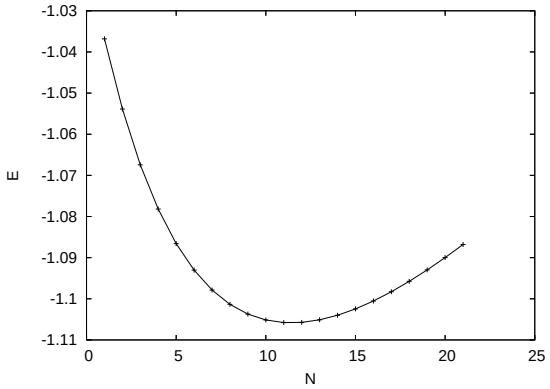

Figure 5.1: First figure

```
          5.4963645520E-03  0.0000000000E+00  0.0000000000E+00
fcart12   6.9585355532E-03  0.0000000000E+00  0.0000000000E+00
         -6.9585355532E-03  0.0000000000E+00  0.0000000000E+00
```

From these two values, using a linear interpolation, one get the optimal value of 1.522 Bohr. Note that *the number of SCF cycles drops from 7 to 5 when the wavefunctions are read from the previous dataset.*

5.1.3 Computation of the interatomic distance (method 2)

1. The other methodology is based on an automatic computation of the minimum. There are different algorithms to do that. See the input variable *ionmov*, with values 2, 3 and 7. In the present case, with only one degree of freedom to be optimized, the best choice is *ionmov* 3.

   You have also to define the maximal number of timesteps for this optimization. Set the input variable "ntime" to 10, it will be largely enough. For the stopping criterion "tolmxf", use the reasonable value of 5.0×10^{-4} Ha/Bohr. This defines the force threshold to consider that the geometry is converged. The code will stop if the residual forces are below that value before reaching "ntime".

   It is also worth to change the stopping criterion for the SCF cycle, in order to be sure that the forces generated for each trial interatomic distance are sufficiently converged. So, change "toldfe" in "toldff", and set the latter input variable to ten times smaller than "tolmxf". The input file `~ABINIT/Tutorial/t13.in` is an example of file that will do the job, while `~ABINIT/Tutorial/Refs/t13.out` is an example of output file. If you decide to use these files, do not forget to change the file names in the `t1x.files` file ... So, you run the code with your input file (it should take less than one minute), examine quietly this file (which is much smaller than the `t12.out` file), and get some significant output data gathered in the final echo of variables:

```
etotal   -1.1058360628E+00
 fcart    1.8438010986E-04  0.0000000000E+00  0.0000000000E+00
         -1.8438010986E-04  0.0000000000E+00  0.0000000000E+00
  ...
 xcart   -7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
          7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
```

According to these data (see *xcart*), the optimal interatomic distance is about 1.520 Bohr, in good agreement with the estimation of `t12.out`. If you have time (this is to be done at

home), you might try to change the stopping criteria, and redo the calculation, to see the level of convergence of the interatomic distance.

Note that the final value of *fcart* in your run might differ slightly from the one shown above (less than one percent change). Such a fluctuation is quite often observed for a value converging to zero (remember, we ask the code to determine the equilibrium geometry, that is, forces should be zero) when the same computation is done on different platforms.

5.1.4 Computation of the charge density

1. We start from the optimized interatomic distance 1.522 Bohr, and make a run at fixed geometry. The input variable "prtden" must be set to 1. To understand correctly the content of the "prtden" description, it is worth to read a much more detailed description of the "files" file, in section 4 of the `abinis_help` file.

2. The input file `~ABINIT/Tutorial/t14.in` is an example of input file for a run that will print a density. If you decide to use this file, do not forget to change the file names in `t1x.files`. The run will take a few seconds.

   The density will be output in the **t1xo_DEN** file. Try to edit it ... No luck! This file is unformatted, not written using the ASCII code. Even if you cannot read it, its description is provided in the `abinis_help`. It contains first a header, then the density numbers. The description of the header is presented in section 6.4 of the `abinis_help` file, while the body of the `_DEN` file in presented in section 6.5. It is the appropriate time to read also the description of the potential files and wavefunctions files, as these files contain the same header as the density file, see sections 6.6 and 6.7.

3. Such a density file can be read by ABINIT, to restart a calculation (see the input variable *iscf*, when its value is -2), but more usually, by an utility called "cut3d". This utility is available in the ABINIT package. You might try to use it now, to generate two–dimensional cuts in the density, and visualize the charge density contours.

   Read the corresponding `cut3d` help file. Then, try to run `cut3d` to analyze `t1xo_DEN`. You should first try to translate the unformatted density data to indexed formatted data, by using option 6 in the adequate menu. Save the indexed formatted data to file `t1xo_DEN_indexed`. Then, edit this file, to have an idea of the content of the `_DEN` files.

   For further treatment, you might choose to select another option than 6. In particular, if you have access to MATLAB, choose option 5. With minor modifications (set *ngx*=*ngy*=*ngz* to 30) you will be able to use the file `dim.m` present in `~ABINIT/Tutorial` to visualize the 3–Dimensional isosurfaces. Another option might be to use the XCrysDen software, for which you need to use option 9.

5.1.5 Computation of the atomization energy

1. The atomization energy is the energy needed to separate a molecule in its constituent atoms, each being neutral. In the present case, one must compute first the total energy of an isolated hydrogen atom. The atomization energy will be the difference between the total energy of H_2 and twice the total energy of H.

   There are some subtleties in the calculation of an isolated atom:

   - in many cases, the ground state of an isolated atom is spin–polarized, see the variables "nsppol" and "spinat";
   - the highest occupied level might be degenerate with the lowest unoccupied level of the same spin, in which case techniques usually appropriate for metals are to be used (see lesson 4);

- also often, the symmetry of the ground–state charge density will NOT be spherical, so that the automatic determination of symmetries by the code, based on the atomic coordinates, should be disabled, see the input variable "nsym", to be set to 1 in this case.

For Hydrogen, we are lucky that the ground state is spherical ($1s$ orbital), and that the highest occupied level and lowest unoccupied level, although degenerate, have a different spin. We will define by hand the occupation of each spin, see the input variables *occopt* (to be set to 2), and *occ*. Finally, in order to make numerical errors cancel, it is important to compute the above–mentioned difference in the same box, for the same cut–off, and even for a location in the box that is similar to the molecule case (although the latter might not be so important).

The input file `~ABINIT/Tutorial/t15.in` is an example of file that will do the job, while `~ABINIT/Tutorial/Refs/t15.out` is an example of output file. If you decide to use the `t15.in` file, do not forget to change the file names in the `t1x.files` file. The run lasts a few seconds.

You should read the output file, and note the tiny differences related with the spin–polarization:

- the electronic eigenvalues are now given for both spin up and spin down cases:

```
Eigenvalues (hartree) for nkpt=   1  k points, SPIN UP:
kpt#   1, nband=  1, wtk=  1.00000, kpt=  0.0000  0.0000  0.0000 (reduced coord)
      -0.26422
Eigenvalues (hartree) for nkpt=   1  k points, SPIN DOWN:
kpt#   1, nband=  1, wtk=  1.00000, kpt=  0.0000  0.0000  0.0000 (reduced coord)
      -0.11117
```

- the spin polarization at each point of the FFT grid is also analyzed:

```
,Min spin pol zeta=  1.0000E+00 at reduced coord.  0.0000  0.0000  0.0000
,         next min=  1.0000E+00 at reduced coord.  0.0333  0.0000  0.0000
,Max spin pol zeta=  1.0000E+00 at reduced coord.  0.9667  0.9667  0.9667
,         next max=  1.0000E+00 at reduced coord.  0.9333  0.9667  0.9667
```

The **zeta** variable is the ratio between the spin–density difference and the charge density. It varies between $+1$ and -1. In the present case of Hydrogen, there is no spin down density, so the zeta variable is $+1$.

The total energy is

```
etotal   -4.7010531340E-01
```

while the total energy of the H_2 molecule is (see test 13):

```
etotal   -1.1058360629E+00
```

The atomization energy is thus 0.1656 Ha.

At this stage, we can compare our results:

- bond length: 1.522 Bohr;
- atomization energy at that bond length: 0.1656 Ha = 4.506 eV;

with the experimental data as well as theoretical data using a much more accurate technique (see Kolos and Roothaan, Rev. Mod. Phys. 32, 219 (1960), especially p.225):

- bond length: 1.401 Bohr;
- atomization energy: 4.747 eV.

The bond length is awful (nearly 10% off), and the atomization energy is a bit too low, 5% off.

What is wrong??

Well, are you sure that the input parameters that we did not discuss are correct? These are:

- *ecut* (the plane–wave kinetic energy cut–off);
- *acell* (the supercell size);
- *ixc* (not even mentioned until now, this input variable specifies what kind of exchange–correlation functional is to be used ...)
- the pseudopotential.

We used 10 Ha as cut–off energy, a $10 \times 10 \times 10$ Bohr^3 supercell, the local–density approximation (as well as the local–spin–density approximation) in the Teter parameterization, and a pseudopotential from the Goedecker–Hutter–Teter table (Phys. Rev. B 54, 1703 (1996)).

We will see in the next lesson how to address the choice of these parameters (except the pseudopotential).

5.1.6 Answers to the questions, section 5.1.1

NOTE: *there might be numerical differences, from platform to platform, in the quoted results!*

1. Q1. 7 SCF cycles were needed:

```
    iter   Etot(hartree)      deltaE(h)  residm     vres2    diffor    maxfor
ETOT  1  -1.1013391225241    -1.101E+00 4.220E-04 8.396E+00 2.458E-02 2.458E-02
ETOT  2  -1.1034123727266    -2.073E-03 4.367E-09 1.668E+00 8.602E-03 3.318E-02
ETOT  3  -1.1037064870489    -2.941E-04 1.836E-05 3.207E-01 4.922E-03 3.810E-02
ETOT  4  -1.1037182046373    -1.172E-05 1.090E-07 8.675E-02 3.620E-04 3.774E-02
ETOT  5  -1.1037224013769    -4.197E-06 1.436E-07 1.829E-04 3.593E-04 3.738E-02
ETOT  6  -1.1037224209642    -1.959E-08 1.123E-09 1.445E-05 3.106E-05 3.741E-02
ETOT  7  -1.1037224213176    -3.534E-10 6.528E-12 8.113E-07 4.102E-06 3.741E-02

At SCF step    7, etot is converged:
 for the second time, diff in etot=  3.534E-10 < toldfe=  1.000E-06
```

2. Q2. Yes, the energy is more converged than *toldfe*, since the stopping criterion asked for the difference between successive evaluations of the energy to be smaller than *toldfe* twice in a row, while the evolution of the energy is nice, and always decreasing by smaller and smaller amounts.

3. Q3. These values are:

```
cartesian forces (hartree/bohr) at end:
   1     -0.03740515236097      0.00000000000000      0.00000000000000
   2      0.03740515236097      0.00000000000000      0.00000000000000
frms,max,avg= 2.1595875E-02 3.7405152E-02   0.000E+00  0.000E+00  0.000E+00 h/b
```

   On the first atom (located at -0.7 0 0 in Cartesian coordinates, in Bohr), the force vector is pointing in the minus x direction, and in the plus x direction for the second atom located at $+0.7$ 0 0.

   The H2 molecule would like to expand ...

4. Q4. The eigenvalues (in Hartree) are mentioned at the lines

```
Eigenvalues (hartree) for nkpt=   1  k points:
kpt#   1, nband=  2, wtk=  1.00000, kpt=  0.0000  0.0000  0.0000 (reduced coord)
 -0.36526  -0.01379
```

As mentioned in the `abinis_help` file, the absolute value of eigenenergies is not meaningful. Only differences of eigenenergies, as well as differences with the potential.

The difference is 0.35147 Hartree, that is 9.564 eV.

Moreover, remember that Kohn–Sham eigenenergies are formally NOT connected to experimental excitation energies!

(Well, more is to be said later about this ...).

5. Q5. The maximum electronic density in electron per Bohr cube is reached at the mid–point between the two H atoms:

```
,Max el dens=  2.6907E-01 el/bohr^3 at reduced coord.  0.0000  0.0000  0.0000
```

5.2 Lesson 2: The H2 molecule, with convergence studies

This lesson aims at showing how to get converged values for the following physical properties:

- the bond length;
- the atomization energy.

You will learn about the numerical quality of the calculations, then make convergence studies with respect to the number of planewaves and the size of the supercell, and finally consider the effect of the XC functional. The problems related to the use of different pseudopotential are left for another lesson (still to be written ...).

You will also finish to read the `abinis_help` file.

This lesson should take about 1 hour to be done.

5.2.1 Summary of the previous lesson

We studied the H_2 molecule in a big box. We used 10 Ha as cut–off energy, a $10 \times 10 \times 10$ Bohr^3 supercell, the local–density approximation (as well as the local–spin–density approximation) in the Teter parameterization (*ixc*=1, the default), and a pseudopotential from the Goedecker–Hutter–Teter table.

At this stage, we compared our results:

- bond length: 1.522 Bohr;
- atomization energy at that bond length: 0.1656 Ha = 4.506 eV.

with the experimental data (as well as theoretical data using a much more accurate technique than DFT):

- bond length: 1.401 Bohr;
- atomization energy: 4.747 eV.

The bond length is awful (nearly 10% off), and the atomization energy is a bit too low, 5% off.

5.2.2 The convergence in ecut

1. **Computing the bond length and corresponding atomization energy in one run.**

   Before beginning, you might consider to work in a different subdirectory as for lesson 1. Why not "Work2"?

   Because we will compute many times the bond length and atomization energy, it is worth to make a single input file that will do all the associated operations. You should try to use 2 datasets (try to combine `~ABINIT/Tutorial/t13.in` with `~ABINIT/Tutorial/t15.in`!). Do not try to have the same position of the H atom as one of the H_2 atoms in the optimized geometry.

   The input file `~ABINIT/Tutorial/t21.in` is an example of file that will do the job, while `~ABINIT/Tutorial/Refs/t21.out` is an example of output file. You might use `~ABINIT/Tutorial/t2x.files` as "files" file (do not forget to modify it), although it does not differ from `~ABINIT/Tutorial/t1x.files`. The run should take less than one minute.

   You should obtain the values:

   ```
   etotal1  -1.1058360629E+00
   etotal2  -4.7010531340E-01
   ```

   and

   ```
   xcart1  -7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
            7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
   ```

   These are similar to those determined in lesson 1, although they have been obtained in one run. You can also check that the residual forces are lower than 5.0×10^{-4}. Convergence issues are discussed in section 7 of the `abinis_help` file.

   By the way, you have read all the most important parts of the `abinis_help` file! You are missing the sections 2, 5, 8. You are also missing the description of many input variables. We suggest that you finish to read entirely the above–mentioned sections of the `abinis_help` file now, while the knowledge of the input variables will come in the long run.

2. Many convergence parameters have been already identified. We will focus only on *ecut* and *acell*. This is because

   - the convergence of the SCF cycle and geometry determination are well under control thanks to *toldfe*, *toldff* and *tolmxf* (this might not be the case for other physical properties);
   - there is no k–point convergence study to be done for an isolated system in a big box: no additional information is gained by adding a k–point beyond one;
   - the *boxcut* value is automatically chosen larger than 2 by ABINIT, see the determination of the input variable "ngfft" by preprocessing;
   - we are using *ionmov*=3 for the determination of the geometry.

   For the check of convergence with respect to *ecut*, you have the choice between doing different runs of the `t21.in` file with different values of *ecut*, or doing a double loop of datasets, as proposed in `~ABINIT/Tutorial/t22.in`. The values of *ecut* have been chosen between 10 Ha and 35 Ha, by step of 5 Ha. If you want to make a double loop, you might benefit of reading again the double–loop section of the `abinis_help` file.

3. You have likely seen a big increase of the CPU time needed to do the calculation (now, a few minutes). You should also look at the increase of the memory needed to do the calculation (go back to the beginning of the output file). The output data are as follows:

```
etotal11 -1.1058360629E+00
etotal12 -4.7010531340E-01
etotal21 -1.1218715957E+00
etotal22 -4.7529731218E-01
etotal31 -1.1291943792E+00
etotal32 -4.7773586216E-01
etotal41 -1.1326879404E+00
etotal42 -4.7899907995E-01
etotal51 -1.1346739190E+00
etotal52 -4.7972721394E-01
etotal61 -1.1359660026E+00
etotal62 -4.8022016187E-01

 xcart11 -7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
          7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
 xcart12  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart21 -7.5104996718E-01  0.0000000000E+00  0.0000000000E+00
          7.5104996718E-01  0.0000000000E+00  0.0000000000E+00
 xcart22  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart31 -7.3977137323E-01  0.0000000000E+00  0.0000000000E+00
          7.3977137323E-01  0.0000000000E+00  0.0000000000E+00
 xcart32  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart41 -7.3304297557E-01  0.0000000000E+00  0.0000000000E+00
          7.3304297557E-01  0.0000000000E+00  0.0000000000E+00
 xcart42  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart51 -7.3001593298E-01  0.0000000000E+00  0.0000000000E+00
          7.3001593298E-01  0.0000000000E+00  0.0000000000E+00
 xcart52  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart61 -7.2955932741E-01  0.0000000000E+00  0.0000000000E+00
          7.2955932741E-01  0.0000000000E+00  0.0000000000E+00
 xcart62  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
```

The corresponding atomization energies and interatomic distances are:

ecut (Ha)	atomisation energy (Ha)	interatomic distance (Bohr)
10	.1656	1.522
15	.1713	1.502
20	.1737	1.480
25	.1747	1.466
30	.1753	1.460
35	.1756	1.459

In order to obtain 0.2% relative accuracy on the bond length or atomization energy, one should use a kinetic cut–off energy of 30 Ha. We will keep in mind this value for the final run.

Well, 30 Ha is a large kinetic energy cut–off! The pseudopotential that we are using for Hydrogen is rather "hard" ...

5.2.3 The convergence in acell

The same technique as for *ecut* should be now used for the convergence in *acell*. We will explore *acell* starting from 8 8 8 to 18 18 18, by step of 2 2 2. We keep *ecut 10* for this study. Indeed, it

is a rather general rule that there is little cross–influence between the convergence of *ecut* and the convergence of *acell*. The file `~ABINIT/Tutorial/t23.in` can be used as an example. The CPU time needed is also in the order of a few minutes. The output data (`~ABINIT/Tutorial/Refs/t23.out`) are as follows:

```
etotal11 -1.1188128742E+00
etotal12 -4.8074164342E-01
etotal21 -1.1058360629E+00
etotal22 -4.7010531340E-01
etotal31 -1.1039109441E+00
etotal32 -4.6767804747E-01
etotal41 -1.1039012761E+00
etotal42 -4.6743724167E-01
etotal51 -1.1041439320E+00
etotal52 -4.6735895144E-01
etotal61 -1.1042058190E+00
etotal62 -4.6736729686E-01

 xcart11 -7.8427119905E-01  0.0000000000E+00  0.0000000000E+00
          7.8427119905E-01  0.0000000000E+00  0.0000000000E+00
 xcart12  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart21 -7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
          7.6091430410E-01  0.0000000000E+00  0.0000000000E+00
 xcart22  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart31 -7.5472620965E-01  0.0000000000E+00  0.0000000000E+00
          7.5472620965E-01  0.0000000000E+00  0.0000000000E+00
 xcart32  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart41 -7.5491934758E-01  0.0000000000E+00  0.0000000000E+00
          7.5491934758E-01  0.0000000000E+00  0.0000000000E+00
 xcart42  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart51 -7.5427689417E-01  0.0000000000E+00  0.0000000000E+00
          7.5427689417E-01  0.0000000000E+00  0.0000000000E+00
 xcart52  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
 xcart61 -7.5415539738E-01  0.0000000000E+00  0.0000000000E+00
          7.5415539738E-01  0.0000000000E+00  0.0000000000E+00
 xcart62  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
```

The corresponding atomization energies and interatomic distances are:

acell (Bohr)	atomization energy (Ha)	interatomic distance (Bohr)
8	.1574	1.568
10	.1656	1.522
12	.1686	1.509
14	.1691	1.510
16	.1694	1.508
18	.1695	1.508

In order to reach 0.2% convergence on the interatomic distance, one needs *acell* 12 12 12. The atomization energy needs *acell* 14 14 14 to be converged at that level. At *12 12 12*, the difference is 0.0009 Ha = 0.024 eV, which is sufficiently small for practical purposes. We will use *acell* 12 12 12 for the final run.

For most solids the size of the unit cell will be smaller than that. We are treating a lot of vacuum in this supercell! So, the H_2 study, with this pseudopotential, turns out to be not really easy. Of course, the number of states to be treated is minimal! This allows to have reasonable CPU time still.

5.2.4 The final calculation in Local (Spin) Density Approximation

We now use the correct values of both *ecut* and *acell*. Well, you should modify the `t23.in` file to make a calculation with *acell* 12 12 12 and *ecut* 30. You can still use the double loop feature with *udtset* 1 2 (which reduces to a single loop), to minimize the modifications to the file. The file `~ABINIT/Tutorial/t24.in` can be taken as an example of input file, and `~ABINIT/Tutorial/Refs/t24.out` as an example of output file.

Since we are doing the calculation at a single (*ecut*, *acell*) pair, the total CPU time is not as much as for the previous determinations of optimal values through series calculations. However, the memory needs have still increased a bit.

The output data are:

```
etotal11 -1.1329372052E+00
etotal12 -4.7765320649E-01

 xcart11 -7.2661954446E-01  0.0000000000E+00  0.0000000000E+00
          7.2661954446E-01  0.0000000000E+00  0.0000000000E+00
 xcart12  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
```

- The corresponding atomization energy is 0.1776 Ha = 4.833 eV;
- The interatomic distance is 1.4532 Bohr
- These are our final data for the local (spin) density approximation.

We have used *ixc*=1 . Other expressions for the local (spin) density approximation *ixc*=2, 3 ... 7 are possible. The values 1, 2, 3 and 7 should give about the same results, since they all start from the XC energy of the homogeneous electron gas, as determined by Quantum Monte Carlo calculations. Other possibilities $ixc = 4, 5, 6$ are older local density functionals, that could not rely on these data.

5.2.5 The use of the Generalized Gradient Approximation

We will use the Perdew–Burke–Ernzerhof functional, proposed in *Phys. Rev. Lett.* **77**, 3865 (1996).

In principle, for GGA, one should use another pseudopotential than for LDA. However, for the special case of Hydrogen, and in general pseudopotentials with a very small core (including only the 1*s* orbital), pseudopotentials issued from the LDA and from the GGA are very similar.

So, we will not change our pseudopotential. This will save us lot of time, as we should not redo an *ecut* convergence test (*ecut* is often characteristic of the pseudopotentials that are used in a calculation).

Independently of the pseudopotential, an *acell* convergence test should not be done again, since the vacuum is treated similarly in LDA or GGA.

So, our final values within GGA will be easily obtained, by setting *ixc* to 11 in the input file `t24.in`. See `~ABINIT/Tutorial/t25.in` for an example.

```
etotal11 -1.1621428502E+00
etotal12 -4.9869631857E-01

 xcart11 -7.1203906739E-01  0.0000000000E+00  0.0000000000E+00
          7.1203906739E-01  0.0000000000E+00  0.0000000000E+00
 xcart12  0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
```

- The corresponding atomization energy is 0.1647 Ha = 4.482 eV;
- The interatomic distance is 1.4241 Bohr;

- These are our final data for the generalized gradient approximation.

Once more, here are the experimental data:

- bond length: 1.401 Bohr;
- atomization energy: 4.747 eV.

In GGA, we are within 2% of the experimental bond length, but 5% of the experimental atomization energy. In LDA, we were within 4% of the experimental bond length, and within 2% of the experimental atomization energy.

Do not forget that the typical accuracy of LDA and GGA varies with the class of materials studied ...

5.3 Lesson 3: Crystalline silicon

This lesson aims at showing you how to get the following physical properties, for an insulator:

- the total energy;
- the lattice parameter;
- the band structure (actually, the Kohn–Sham band structure).

You will learn about the use of k–points, as well as the smearing of the planewave kinetic energy cut–off.

This lesson should take about 1 hour to be done.

5.3.1 Computing the total energy of silicon at fixed number of k–points

Before beginning, you might consider to work in a different subdirectory as for lesson 1 or lesson 2 . Why not "Work3"?

The file `~ABINIT/Tutorial/t3x.files` lists the file names and root names. You can copy it in the `Work3` directory and change it as you did for the `t1x.files` and `t2x.files` files. You can also copy the file `~ABINIT/Tutorial/t31.in` in Work3. This is your input file. You should edit it, read it carefully, have a look at the following "new" input variables, and their explanation:

- *rprim*;
- *xred* (used instead of *xcart*);
- *kptopt*, *ngkpt*, *nshiftk*, *shiftk*, *kptrlatt* (not easy ... take your time!);
- *diemac* (compared to isolated molecules, another value is used, while *diemix* has been suppressed).

Note also the following: you will work at fixed *ecut* (=8Ha). It is implicit that in "real life", you should do a convergence test with respect to *ecut* ... Here, a suitable *ecut* is given to you. It will allow to obtain 0.2% relative accuracy on lattice parameters.

When you have read the input file, you can run the code, as usual (it will run a few seconds). Then, read the output file, and note the total energy.

```
etotal   -8.8662238960E+00
```

5.3.2 Starting the convergence study with respect to k–points

There is of course a convergence study associated to the sampling of the Brillouin zone. You should examine different grids, of increasing resolution. You might try the following series of grids:

```
ngkpt1  2 2 2
ngkpt2  4 4 4
ngkpt3  6 6 6
ngkpt4  8 8 8
```

However, the associated number of k–points in the irreducible Brillouin zone grows very fast. It is

```
nkpt1  2
nkpt2 10
nkpt3 28
nkpt4 60
```

ABINIT computes this number of k–point, from the definition of the grid and the symmetries. You might define an input *nkpt* value, in which case ABINIT will compare its computed value with this input value. We take this opportunity to examine the behavior of ABINIT when a problem is detected. Let's suppose that with *ngkpt1* 4 4 4, one mentions *nkpt1* 2. The input file `~ABINIT/Tutorial/t32.in` is an example. Do not forget to change `t3x.files`, if you are using that file name. The message that you get at the end of the log file is:

```
inkpts : ERROR -
 The input value of nkpt=     2, does not match
 the number of k points generated by kptopt, ngkpt, shiftk,
 ane the eventual symmetries, that is, nkpt=    10.
 Action : change nkpt in your input file, or one of the other input variables.
```

This is a typical ABINIT error message. It announce clearly that you should use *nkpt* 10.

As the computation of *nkpt* for specific grids of k–points is not an easy task, while the even more important selection of specific economical grids (the best ratio between the accuracy of the integration in the Brillouin zone and the number of k–points) is more difficult, some help to the user is provided by ABINIT. ABINIT is able to examine automatically different k–point grids, and to propose the best grids for integration. This is described in the `abinis_help` file, see the input variable *prtkpt*, and the associated characterization of the integral accuracy, described in *kptrlen*. The generation of lists of k–point sets is done in different test cases, in the directory `Test_v2`. You can directly have a look at the output files in `~ABINIT/Test_v2/Refs`, the output files for the tests 61 to 73.

When one begins the study of a new material, it is strongly advised to examine first the list of k–points grids, and select (at least) three efficient ones, for the k–point convergence study. Do not forget that the CPU time will be linearly proportional to the number of k–points to be treated: using 10 k–points will take five more time than using 2 k–points. Even for a similar accuracy of the Brillouin zone integration (about the same value of *kptrlen*), it might be easy to generate a grid that will fold to 10 k in the irreducible Brillouin zone, as well as one that will fold to 2 k–points in the irreducible Brillouin zone. The latter is clearly to be preferred!

5.3.3 Actually performing the convergence study with respect to k–points

In order to understand k–point grids, you should read the Monkhorst and Pack paper, *Phys. Rev. B* **13**, 5188 (1976) ... Well, maybe not immediately ... In the meantime, you can try the above–mentioned convergence study.

The input file `~ABINIT/Tutorial/t33.in` is an example, while `~ABINIT/Tutorial/Refs/t33.out` is a reference output file. In this output file, you should have a look at the echo of input

variables. As you know, these are preprocessed, and, in particular, *ngkpt* and *shiftk* are used to generate the list of k–points (*kpt*) and their weights (*wtk*). You should read the information about *kpt* and *wtk*.

From the output file, here is the evolution of total energy per unit cell:

```
etotal1  -8.8662238960E+00
etotal2  -8.8724909739E+00
etotal3  -8.8726017429E+00
etotal4  -8.8726056405E+00
```

The difference between dataset 3 and dataset 4 is rather small. Even the dataset 2 gives an accuracy of about 0.0001 Ha. So, our converged value for the total energy, at fixed *acell*, fixed *ecut*, is -8.872 Ha.

5.3.4 Determination of the lattice parameters

The input variable "optcell" governs the automatic optimization of cell shape and volume. For the automatic optimization of cell volume, use:

```
opcell 1
ionmov 3
ntime 10
dilatmx 1.05
ecutsm 0.5
```

You should read the indications about *dilatmx* and *ecutsm*. Do not test all the k–point grids, only those with *nkpt* 2 and 10.

The input file `~ABINIT/Tutorial/t34.in` is an example, while `~ABINIT/Tutorial/Refs/t34.out` is a reference output file.

This run might last a few minutes ...

You should obtain the following evolution of the lattice parameters:

```
acell1   1.0230001904E+01  1.0230001904E+01  1.0230001904E+01 Bohr
acell2   1.0216682464E+01  1.0216682464E+01  1.0216682464E+01 Bohr
```

with the following very small residual stresses:

```
strten1  -2.5365388633E-08 -2.5365388633E-08 -2.5365388633E-08
          0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
strten2  -5.3567080431E-08 -5.3567080431E-08 -5.3567080431E-08
          0.0000000000E+00  0.0000000000E+00  0.0000000000E+00
```

The stress tensor is given in Hartree/Bohr^3, and the order of the components is

```
11  22  33
23  13  12
```

There is only a 0.13% relative difference between *acell1* and *acell2*. So, our converged LDA value for Silicon, with the `14si.pspnc` pseudopotential (see the `t3x.files` file) is 10.217 Bohr, that is 5.407 Angstrom. The experimental value is 5.431 Angstrom at 25 degree Celsius, see R. W. G. Wyckoff, *Crystal structures*, Ed. Wiley and sons, New–York (1963).

5.3.5 Computing the band structure

We fix the parameters *acell* to the theoretical value of 3×10.217, and we fix also the grid of k–points (the $4 \times 4 \times 4$ FCC grid, equivalent to a $8 \times 8 \times 8$ Monkhorst–pack grid).

We will ask for 8 bands (4 valence and 4 conduction).

A band structure can be computed by solving the Kohn–Sham equation for many different k–points, along different lines of the Brillouin zone. The potential that enters the Kohn–Sham must be derived from a previous self–consistent calculation, and will not vary during the scan of different k–point lines.

Suppose that you want to make a L–Gamma–X–(U–)Gamma circuit, with 10, 12 and 17 divisions for each line (each segment has a different length in reciprocal space, and these divisions give approximately the same distance between points along a line). The circuit will be obtained easily by the following choice of segment end points:

```
L     (1/2 0 0)
Gamma (0 0 0)
X     (0 1/2 1/2)
Gamma (1 1 1)
```

Note:

1. the last Gamma point is in another cell of the reciprocal space than the first one, this choice allows to construct the X–U–Gamma line easily;
2. the k–points are specified using reduced coordinates — in agreement with the input setting of the primitive 2–atom unit cell — in standard textbooks, you will often find the L, Gamma or X point given in coordinates of the conventional 8–atom cell: the above–mentioned circuit is then (1/2 1/2 1/2)–(0 0 0)–(1 0 0)–(1 1 1), but such coordinates cannot be used with the 2–atom cell.

So, you should set up in your input file, for the first dataset, a usual SCF calculation in which you output the density (*prtden* 1), and, for the second dataset:

- fix *iscf* to -2, to make a non–self–consistent calculation;
- define *getden* -1, to take the output density of dataset 1;
- set *nband* to 8;
- set *kptopt* to -3, to define three segments in the Brillouin Zone;
- set *ndivk* to 10 12 17 (this means a circuit defined by 4 points, with 10 divisions of the first segment, 12 divisions of the second, 17 divisions of the third);
- set *kptbounds* to

```
0.5  0.0  0.0 # L point
0.0  0.0  0.0 # Gamma point
0.0  0.5  0.5 # X point
1.0  1.0  1.0 # Gamma point in another cell.
```

- set *enunit* to 1, in order to have eigenenergies in eV;
- the only tolerance criterion admitted for non–self–consistent calculations is *tolwfr*. You should set it to 1.0×10^{-10} (or so), and suppress *toldfe*.

The input file `~ABINIT/Tutorial/t35.in` is an example, while `~ABINIT/Tutorial/Refs/t35.out` is a reference output file.

You should find the band structure starting at (second dataset):

```
Eigenvalues (   eV  ) for nkpt=  40  k points:
kpt#   1, nband=  8, wtk=  1.00000, kpt=  0.5000  0.0000  0.0000 (reduced coord)
 -3.78987  -1.16032   4.69394   4.69394   7.38389   9.23579   9.23579  13.45363
kpt#   2, nband=  8, wtk=  1.00000, kpt=  0.4500  0.0000  0.0000 (reduced coord)
```

```
 -3.92925  -0.95943   4.71018   4.71018   7.40286   9.25271   9.25271  13.48581
kpt#   3, nband=  8, wtk=  1.00000, kpt=  0.4000  0.0000  0.0000 (reduced coord)
 -4.25584  -0.44579   4.76451   4.76451   7.46440   9.30899   9.30899  13.57381
kpt#   4, nband=  8, wtk=  1.00000, kpt=  0.3500  0.0000  0.0000 (reduced coord)
 -4.64158   0.24736   4.85456   4.85456   7.56450   9.38020   9.38020  13.64227
....
```

One needs a graphical tool to represent all these data ... (For the MAPR 2451 lecture: try with MATLAB)

Even without a graphical tool we will have a quick look at the values at L, Gamma, X and Gamma again:

```
kpt#   1, nband=  8, wtk=  1.00000, kpt=  0.5000  0.0000  0.0000 (reduced coord)
 -3.78987  -1.16032   4.69394   4.69394   7.38389   9.23579   9.23579  13.45363

kpt#  11, nband=  8, wtk=  1.00000, kpt=  0.0000  0.0000  0.0000 (reduced coord)
 -6.17102   5.91515   5.91515   5.91515   8.44524   8.44524   8.44524   9.17125

kpt#  23, nband=  8, wtk=  1.00000, kpt=  0.0000  0.5000  0.5000 (reduced coord)
 -1.96598  -1.96598   3.00347   3.00347   6.50928   6.50928  15.94976  16.44101

kpt#  40, nband=  8, wtk=  1.00000, kpt=  1.0000  1.0000  1.0000 (reduced coord)
 -6.17102   5.91515   5.91515   5.91515   8.44524   8.44524   8.44524   9.17125
```

The last gamma is exactly equivalent to the first gamma. It can be checked that the top of the valence band is obtained at Gamma (=15.56202 eV). The width of the valence band is 12.09 eV, the lowest unoccupied state at X is 0.594 eV higher than the top of the valence band, at Gamma. The Si is described as an indirect band gap material (this is correct), with a band–gap of about 0.594 eV (this is quantitatively quite wrong: the experimental value 1.17 eV is at 25 degree Celsius). The minimum of the conduction band is even slightly displaced with respect to X, see *kpt* # 21 . This underestimation of the band gap is well–known (the famous DFT band–gap problem). In order to obtain correct band gaps, you need to go beyond the Kohn–Sham Density Functional Theory: use the GW approximation. This is described in the sixth lesson of the tutorial.

For experimental data and band structure representation, see M. L. Cohen and J. R. Chelikowski, *Electronic structure and optical properties of semiconductors*, Springer–Verlag New–York (1988).

There is a subtlety that is worth to comment about. In non–self–consistent calculations, like those performed in the present band structure calculation, with *iscf* $= -2$, not all bands are converged within the tolerance *tolwfr*. Indeed, the two upper bands (by default) have not been taken into account to apply this convergence criterion: they constitute a "buffer". The number of such "buffer" bands is governed by the input variable *nbdbuf*.

It can happen that the highest (or two highest) band(s), if not separated by a gap from non–treated bands, can exhibit a very slow convergence rate. This buffer allows to achieve convergence of "important", non–buffer bands. In the present case, 6 bands have been converged with a residual better than *tolwfr*, while the two upper bands are less converged (still sufficiently for graphical representation of the band structure). In order to achieve the same convergence for all 8 bands, it is advised to use *nband* $= 10$ (that is, $8 + 2$).

5.4 Lesson 4: Aluminum, the bulk and the surface

This lesson aims at showing how to get the following physical properties, for a metal, and for a surface:

- the total energy;

- the lattice parameter;
- the relaxation of surface atoms;
- the surface energy.

You will learn about the smearing of the Brillouin zone integration, and also a bit about preconditioning the SCF cycle.

This lesson should take about 1 hour and 30 minutes to be done.

5.4.1 Computing the total energy and lattice parameters of aluminum for a fixed smearing and number of k–points.

Before beginning, you might consider to work in a different subdirectory as for lesson 1, 2 or 3 . Why not "Work4"?

The file `~ABINIT/Tutorial/t4x.files` lists the file names and root names. You can copy it in the Work4 directory (and change it, as usual). You can also copy the file `~ABINIT/Tutorial/t41.in` in Work4. This is your input file. You should edit it, read it carefully, and have a look at the following "new" input variables, and their explanation:

- *occopt*;
- *tsmear*.

Note also the following:

1. You will work at fixed *ecut* (=6Ha). It is implicit that in "real life", you should do a convergence test with respect to *ecut* ... Here, a suitable *ecut* is given to you. It will allow to obtain 0.2% relative accuracy on lattice parameters. Note that this is the softer pseudopotential of those that we have used until now: the `01h.pspgth` for H needed 30 Ha (it was rather hard), the `14si.pspnc` for Si needed 8 Ha.
2. The input variable *diemac* has been suppressed. Aluminum is a metal, and the default is taylored for that case.

When you have read the input file, you can run the code, as usual (it will take a few seconds). Then, read the output file quietly. You should note that the Fermi energy and occupation numbers have been computed automatically:

```
Fermi energy (hartree) =   0.26800
Eigenvalues (hartree) for nkpt=   2  k points:
kpt#   1, nband=  3, wtk=  0.75000, kpt= -0.2500  0.5000  0.0000 (reduced coord)
  0.09391   0.25391   0.41846
     occupation numbers for kpt#   1
  2.00003   1.33306   0.00014
kpt#   2, nband=  3, wtk=  0.25000, kpt= -0.2500  0.0000  0.0000 (reduced coord)
 -0.07058   0.41033   0.68787
     occupation numbers for kpt#   2
  2.00000   0.00030   0.00000
```

You should also note that the components of the total energy include an entropy term:

```
Eight components of total free energy (hartree) are
kinetic= 8.70148725738375E-01 Hartree= 3.84240572509002E-03
     xc=-8.08093663953667E-01 loc psp= 8.21205082511040E-02
nl psp = 4.52424282437438E-01 pspcore= 3.78200676875296E-02
-kT*entropy =-7.99762559838938E-03 (internal energy=-2.08996839059198E+00 )
Ewald  =-2.72823071647785E+00 resulting in Etotal=-2.09796601619037E+00 hartree
Also Etotal= -5.70885576267268E+01 eV ; Eeig (band energy)=  3.5951203776E-01 Ha
```

5.4.2 The convergence study with respect to k–points

There is of course a convergence study associated to the sampling of the Brillouin zone. You should examine different grids, of increasing resolution. You might try the following series of grids:

```
ngkpt1  2 2 2
ngkpt2  4 4 4
ngkpt3  6 6 6
ngkpt4  8 8 8
```

with the associated *nkpt*:

```
nkpt1  2
nkpt2 10
nkpt3 28
nkpt4 60
```

The input file `~ABINIT/Tutorial/t42.in` is an example, while `~ABINIT/Tutorial/Refs/t42.out` is a reference output file. The run might take more than one minute.

You will see that, FOR THE PARTICULAR VALUE OF *tsmear* = 0.05 Ha, the lattice parameter is already converged with *nkpt* = 10:

```
acell1   7.5623662498E+00  7.5623662498E+00  7.5623662498E+00 Bohr
acell2   7.5084285443E+00  7.5084285443E+00  7.5084285443E+00 Bohr
acell3   7.5013992940E+00  7.5013992940E+00  7.5013992940E+00 Bohr
acell4   7.4990383260E+00  7.4990383260E+00  7.4990383260E+00 Bohr
```

Note that there is usually a STRONG cross–convergence effect between the number of k–points and the value of the broadening, *tsmear*. The right procedure is: for each value of *tsmear*, get the convergence with respect with the number of k–points, then to compare the k–point converged values for different values of *tsmear*.

In what follows, we will restrict ourselves to the grids with *nkpt*=2, 10 and 28.

5.4.3 The convergence study with respect to both number of k–points AND broadening factor (tsmear)

The theoretical convergence rate for *tsmear* ending to 0, in the case of *occopt* = 4, is quartic. This is obtained in the hypothesis of infinitely dense k–point grid. We will check the evolution of *acell* as a function of *tsmear*, for the following values of *tsmear*: 0.01, 0.02, 0.03 and 0.04. Use the double–loop capability of the multi–dataset mode, with the *tsmear* changes in the INNER loop. This will saves CPU time, as the wavefunctions of the previous dataset will be excellent (no transfer to different k–points).

The input file `~ABINIT/Tutorial/t43.in` is an example, while `~ABINIT/Tutorial/Refs/t43.out` is a reference output file.

From the output file, here is the evolution of *acell*:

```
acell11  7.5622298688E+00  7.5622298688E+00  7.5622298688E+00 Bohr
acell12  7.5622412743E+00  7.5622412743E+00  7.5622412743E+00 Bohr
acell13  7.5622412745E+00  7.5622412745E+00  7.5622412745E+00 Bohr
acell14  7.5622427067E+00  7.5622427067E+00  7.5622427067E+00 Bohr
acell21  7.5071087625E+00  7.5071087625E+00  7.5071087625E+00 Bohr
acell22  7.5071970473E+00  7.5071970473E+00  7.5071970473E+00 Bohr
acell23  7.5032517079E+00  7.5032517079E+00  7.5032517079E+00 Bohr
acell24  7.5055911048E+00  7.5055911048E+00  7.5055911048E+00 Bohr
acell31  7.4958511018E+00  7.4958511018E+00  7.4958511018E+00 Bohr
acell32  7.4952121945E+00  7.4952121945E+00  7.4952121945E+00 Bohr
acell33  7.4965135656E+00  7.4965135656E+00  7.4965135656E+00 Bohr
acell34  7.4990025833E+00  7.4990025833E+00  7.4990025833E+00 Bohr
```

These data should be analyzed properly.

For *tsmear* = 0.01, the converged value, contained in *acell31*, must be compared to *acell11* and *acell21*: between *acell21* and *acell31*, the difference is below 0.2%. *acell31* can be considered to be converged with respect to the number of k–points, at fixed *tsmear*. This *tsmear* being the lowest one, it is usually the most difficult to converge, and the values *acell31,32,33* and *34* are indeed well–converged with respect to the k–point number.

The use of the largest *tsmear*(=0.04), giving *acell34*, induces only a small error in the lattice parameter. For that particular value of *tsmear*, one can use the second k–point grid, giving *acell24*.

So to **summarize**:

We can choose to work with a 10 k–point grid in the irreducible Brillouin zone, and the associated $tsmear = 0.04$, with less than 0.1% error on the lattice parameter.

NOTE that this error due to the Brillouin zone sampling could add to the error due to the choice of *ecut* (that was mentioned previously to be on the order of 0.2%).

In what follows, we will stick to these values of *ecut* and *tsmear*, and try to use k–point grids with a similar resolution.

Our final value for the aluminum lattice parameter, in the LDA, using the `13al.981214.fhi` pseudopotential, is thus 7.5056 Bohr, that is 3.9718 Å. The experimental value at 25 degree Celsius is 4.04958 Å.

The associated total energy and accuracy can be deduced from

```
etotal11 -2.0915880134E+00
etotal12 -2.0931821220E+00
etotal13 -2.0947762307E+00
etotal14 -2.0963703493E+00
etotal21 -2.0969479910E+00
etotal22 -2.0975288692E+00
etotal23 -2.0977992413E+00
etotal24 -2.0979739819E+00
etotal31 -2.0983273553E+00
etotal32 -2.0982967240E+00
etotal33 -2.0983057844E+00
etotal34 -2.0983969839E+00
```

etotal24 is -2.0979739819E+00 Ha, with an accuracy of 0.0005 Ha.

5.4.4 Determination of the surface energy of aluminum (100): changing the orientation of the unit cell

In order to study the Aluminum (100) surface, we will have to set up a supercell representing a slab. This supercell should be chosen as to be compatible with the primitive surface unit cell.

The corresponding directions are [-1 1 0] and [1 1 0]. The direction perpendicular to the surface is [0 0 1]. There is no primitive cell of bulk aluminum based on these vectors, but a doubled cell. We will first compute the total energy associated with this doubled cell. This is not strictly needed, but it is a valuable intermediate step towards the study of the surface.

You might start from `t43.in`.

You have to change *rprim*. Still, try to keep *acell* at the values of bulk aluminum that were determined previously. But it is not all: the most difficult part in the passage to this doubled cell is the definition of the k–point grid. Of course, one could just take a homogeneous simple cubic grid of k–points, but this will not correspond exactly to the k–point grid used in the primitive cell in `t43.in`. This would not be a big problem, but you would miss some error cancellation.

The answer to this problem is given in the input file `~ABINIT/Tutorial/t44.in`. The procedure to do the exact translation of the k–point grid will not be explained here (sorry for this). If you do not see how to do it, just use homogeneous simple cubic grids, with about the same resolution as for the primitive cell case. There is a simple rule to estimate ROUGHLY whether

two grids for different cells have the same resolution: simply multiply the linear dimensions of the k–point grids, by the number of sublattices, by the number of atoms in the cell. For example, the corresponding product for the usual 10 k–point grid is 4x4x4 x 4 x 1 = 256 . In the file `t44.in`, one has 4x4x4 x 2 x 2 = 256 . The grids of k–points should not be too anisotropic for this rough estimation to be valid.

Note also the input variables *rprim* and *chkprim* in this input file.

So, you run `t44.in` (only a few seconds, the reference file is `~ABINIT/Tutorial/Refs/t44.out`), and you find the following total energy:

```
etotal    -4.1962972596E+00
```

It is not exactly twice the total energy for the primitive cell, mentioned above, but the difference is less than 0.0005 Ha. It is due to the different FFT grids used in the two runs, and affect the exchange–correlation energy. These grids are always homogeneous primitive 3D grids, so that changing the orientation of the lattice will give mutually incompatible lattices. Increasing the size of the FFT grid would improve the agreement.

5.4.5 Determination of the surface energy: a (3 aluminum layer + 1 vacuum layer) slab calculation

We will first compute the total energy associated with only three layers of aluminum, separated by only one layer of vacuum. This is kind of a minimal slab:

- one surface layer;
- one "bulk" layer;
- one surface layer;
- one vacuum layer;
- ...

It is convenient to take the vacuum region as having a multiple of the width of the aluminum layers, but this is not mandatory. The supercell to use is the double of the previous cell (that had two layers of Aluminum atoms along the [0 0 1] direction). Of course, the relaxation of the surface might give an important contribution to the total energy.

You should start from `t44.in`.

You have to modify *rprim* (double the cell along [0 0 1]), the atomic positions, as well as the k–point mesh. For the latter, it is supposed that the electrons cannot propagate from one slab to its image in the [0 0 1] direction, so that the k_z component of the special k–points can be taken 0: only one layer of k–points is needed along the z–direction. You should also allow the relaxation of atomic positions, but not the relaxation of lattice parameters (the lattice parameters along x or y must be considered fixed to the bulk value, while, for the z direction, there is no interest to allow the vacuum region to collapse!

The input file `~ABINIT/Tutorial/t45.in` is an example, while `~ABINIT/Tutorial/Refs/t45.out` is a reference output file. The run might last one minute.

The total energy after the first SCF cycle, when the atomic positions are equal to their starting values, is:

```
ETOT  7  -6.2619731934699
```

Note that the total energy of three aluminum atoms in the bulk, (from section 4.3, *etotal24*) is

```
-6.293922 Ha
```

So that the non–relaxed surface energy, per surface unit cell (there are two surfaces in our simulation cell!) is

```
0.015975 Ha = 0.435 eV .
```

The total energy after the Broyden relaxation is:

```
etotal    -6.2622233982E+00
```

so that the relaxed surface energy, per surface unit cell is

```
0.015849 Ha = 0.431 eV .
```

It seems that the relaxation energy is very small, compared to the surface energy, but we need to do the convergence studies.

5.4.6 Determination of the surface energy: increasing the number of vacuum layers

One should now increase the number of vacuum layers: 2 and 3 layers instead of only 1. It is preferable to define atomic positions in cartesian coordinates. The same coordinates will work for both 2 and 3 vacuum layers, while this is not the case for reduced coordinates, as the cell size increases.

The input file `~ABINIT/Tutorial/t46.in` is an example input file, while `~ABINIT/Tutorial/Refs/t46.out` is a reference output file. The run might take a few minutes ...

In the Broyden step 0 of the first dataset, you will notice the WARNING:

```
scprqt:  WARNING -
 nstep=   10 was not enough SCF cycles to converge;
 maximum force difference=  1.716E-04 exceeds toldff=  5.000E-05
```

The SCF convergence is indeed getting more difficult. This is because the default preconditioner (see the notice of the input variable *dielng*) is not very good for the metal+vacuum case.

For the interpretation of the present run, this is not critical, as the convergence criterion was close of being fulfilled, but one should keep this in mind, as you will see ...

For the 2 vacuum layer case, one has the non–relaxed total energy:

```
ETOT 10  -6.2538519290781
```

(that is inaccurate at the 1.0d-4Ha level) giving the unrelaxed surface energy

```
0.0200 Ha = 0.544 eV ;
```

and for the relaxed case:

```
etotal1  -6.2546977224E+00
```

(this one is converged to the required level) giving the relaxed surface energy

```
0.0196 Ha = 0.533 eV
```

Note that the difference between unrelaxed and relaxed case is a bit larger than in the case of one vacuum layer. This is because there was some interaction between slabs of different supercells.

For the 3 vacuum layer case, the self–consistency problem becomes even more severe than with 2 vacuum layers! The Broyden steps 0 and 1 are NOT sufficiently converged (one might set *nstep* to a larger value, but the best is to change the preconditioner, as described below) ...

However, for the Broyden steps number 2 and beyond, because one takes advantage of the previous wavefunctions, a sufficient convergence is reached. The total energy, in the relaxed case, is:

```
total2  -6.2559056529E+00
```

giving the relaxed surface energy 0.0190 Ha = 0.515 eV. There is a rather small 0.018 eV difference with the 2 vacuum layer case.

For the next run, we will keep the 2 vacuum layer case, and we know that the accuracy of the coming calculation cannot be better than 0.016 eV. One might investigate the 4 vacuum layer case, but this is not worth, in the present tutorial ...

5.4.7 Determination of the surface energy: increasing the number of aluminum layers

One should now increase the number of aluminum layers, while keeping 2 vacuum layers. We will consider 4 and 5 aluminum layers. This is rather straightforward to set up, but the problem with the preconditioner is more embarrassing. One could use an effective dielectric constant of about 3 or 5, with a rather small mixing coefficient, on the order of 0.2. However, there is also another possibility, using an estimation of the dielectric matrix governed by *iprcel* = 45. For comparison with the previous treatment of SCF, one can recompute the result with 3 aluminum layers.

The input file `~ABINIT/Tutorial/t47.in` is an example, while `~ABINIT/Tutorial/Refs/t47.out` is a reference output file. This run might take a few minutes, and is the longer of the tutorial. You should start it now.

You can monitor its evolution by editing from time to time the `t47_STATUS` file that the code updates regularly. The status file, that refer to the skeleton of the code, is described in the `~ABINIT/Infos/Notes_for_coding/programmer_guide`. You might take advantage of the time of the run to explore the files contained in the `~ABINIT/Infos` directory and the `~ABINIT/Infos/Notes_for_coding` directory. The `README` files provided interesting entry points in the documentation of the code.

Coming back to the file `t47.out` ...

You will notice that the SCF convergence is now excellent, for all the cases (3, 4 or 5 metal layers).

For the 3 aluminum layer case, one has the non–relaxed total energy:

```
ETOT  7  -6.2539524354404
```

(this quantity is converged, unlike in test 4.6) giving the unrelaxed surface energy 0.0200 Ha = 0.544 eV; and for the relaxed case:

```
etotal1  -6.2547004716E+00
```

(by contrast the difference with test 4.6 is less than 1 microHa) giving the relaxed surface energy 0.0196 Ha = 0.533 eV.

For the 4 aluminum layer case, one has the non–relaxed total energy:

```
ETOT  8  -8.3546873347493
```

giving the unrelaxed surface energy 0.0186Ha = 0.506 eV; and for the relaxed case:

```
etotal2  -8.3565574035E+00
```

giving the relaxed surface energy 0.0183 Ha = 0.498 eV.

For the 5 aluminum layer case, one has the non–relaxed total energy:

```
ETOT  8  -10.453642176501
```

giving the unrelaxed surface energy 0.0183Ha = 0.498 eV; and for the relaxed case:

```
etotal3  -1.0454163549E+01
```

giving the relaxed surface energy 0.0180 Ha = 0.490 eV.

The relative difference in the surface energy of the 4 and 5 layer cases is on the order of 1.5%.

In the framework of this tutorial, we will not pursue this investigation, which is a simple application of the concepts already explored.

Just for your information, and as an additional warning, when the work accomplished until now is completed with 6 and 7 layers without relaxation (see `~ABINIT/Tutorial/t48.in` and

~ABINIT/Tutorial/Refs/t48.out where 5, 6 and 7 layers are treated), this non–relaxed energy surface energy behaves as follows:

number of aluminum layers	surface energy
3	0.544 eV
4	0.506 eV
5	0.498 eV
6	0.449 eV
7	0.463 eV

So, the surface energy convergence is rather difficult to reach.

Our values, with a 4x4x1 grid, a smearing of 0.04 Ha, a kinetic energy cut–off of 6 Ha, the 13al.981214.fhi pseudopotential, still oscillate between 0.45 eV and 0.51 eV.

An error on the order of 0.016 eV is due to the thin vacuum layer. Other sources of errors might have to be rechecked, seeing the kind of accuracy that is needed.

Experimental data give a surface energy around 0.55 eV (sorry, the reference is to be provided).

5.5 Lesson 5: Dynamical and dielectric properties of AlAs

This lesson aims at showing how to get the following physical properties, for an insulator:

- the phonon frequencies and eigenvectors at Gamma;
- the dielectric constant;
- the Born effective charges;
- the LO–TO splitting;
- the phonon frequencies and eigenvectors at other q–points in the Brillouin Zone;
- the interatomic force constants (not yet);
- the phonon band structure from interatomic force constants (not yet);
- associated thermodynamical properties (not yet).

You will learn to use of response–function features of ABINIT. In a future version, you will learn the use of the associated codes Mrgddb and Anaddb.

This lesson should take about (to be provided) hours to be done.

5.5.1 The ground–state geometry of AlAs

Before beginning, you might consider to work in a different subdirectory as for the other lessons. Why not "Work5"?

The file ~ABINIT/Tutorial/t5x.files lists the file names and root names. You can copy it in the Work5 directory (and change it, as usual). Note that two pseudopotentials are mentioned in this "files" file: one for the Aluminum atom, and one for the Arsenic atom. The first to be mentioned, for Al, will define the first type of atom. The second to be mentioned, for As, will define the second type of atom. It is the first time that you encounter this situation in the tutorials.

You can also copy the file ~ABINIT/Tutorial/t51.in in Work5. This is your input file. You should edit it, read it carefully, and, because of the use of two types of atoms, have a look at the following input variables:

- *ntypat*;
- *typat*.

Note that the value of *tolvrs* is rather stringent. This is because the wavefunctions determined by the present run will be used later as starting point of the response–function calculation.

You will work at fixed *ecut* (=3Ha) and k–point grid, defined by *kptrlatt* (the 4x4x4 Monkhorst–Pack grid). It is implicit that in "real life", you should do a convergence test with respect to both convergence parameters. We postpone the discussion of the accuracy of these choices and the choice of pseudopotential to the last section of this tutorial (LINK TO BE GIVEN). They give acceptable results, not very accurate, but, more important, the speed is reasonable for a tutorial.

You should make the run (a dozen of second on a PIII at 450 MHz), and obtain the following value for the energy, in the final echo section:

```
etotal   -9.7626837450E+00
```

However, we will rely later on a more accurate (more digits) value of this total energy, that can be found about a dozen of lines before this final echo:

```
Ewald  =-8.47989583509473E+00 resulting in Etotal=-9.76268374500102E+00 hartree
```

The output file also mention that the forces on both atoms vanish.

The run that you just made will be considered as defining a ground–state configuration, on top of which response functions will be computed. The main output of this ground–state run is the wavefunction file `t51_oWFK`, that you can already rename as `t51_iWFK`. Indeed, it will be used in the next runs, as an input file. So, in the corresponding "files" file, third line, pay attention to specify "t51_i", even if you change the root name for output files (fourth line) to "t52_o" or "t53_o" ...

5.5.2 Frozen–phonon calculation of a second derivative of the total energy

We will now aim at computing the second derivative of the total energy with respect to an atomic displacement by different means. For that purpose, you must first read section 0 and the first paragraph of section 1 of the `respfn_help` file (the auxiliary help file, that deals specifically with the response function features).

We will explain later, in more detail, the signification of the different input parameters introduced in section 1 of the `respfn_help` file.

For now, in order to be able to perform a direct comparison with the result of a response–function calculation, we choose as a perturbation the displacement of the Al atom along the first axis of the reduced coordinates.

You can copy the file `~ABINIT/Tutorial/t52.in` in Work5. This is your input file. You should edit it and briefly look at the two changes with respect to the file `~ABINIT/Tutorial/t51.in`: the change of *xred*, and the reading of the wavefunction file, using the *irdwfk* input variable. Then, you can make the run. The symmetry is lowered with respect to the ground–state geometry, so that the number of k–points increases a lot, and of course, the CPU time (about one minute on a PIII 450 MHz).

From this run, it is possible to get the values of the total energy, and the value of the gradient of the total energy with respect to change of reduced coordinate:

```
rms dE/dt=  3.5517E-03; max dE/dt=  5.0079E-03; dE/dt below (all hartree)
   1       0.005007930232      0.002526304574      0.002526304574
   2      -0.005007868293     -0.002526274885     -0.002526274885
...
Ewald  =-8.47988991313938E+00 resulting in Etotal=-9.76268124105590E+00 hartree
```

The change of reduced coordinate of the Al atom along the first axis was rather small (1/1000), and we can make an estimate of the second derivative of the total energy with respect to the reduced coordinate thanks to finite–difference formulas.

We start first from the total energy difference. The total energy is symmetric with respect to that perturbation, so that it has no linear term. The difference between the ground–state value (-9.76268374500102E+00 hartree) of the previous run, and the perturbed value (-9.76268124105590E+00 hartree) of the present one, is thus one half of the square of the coordinate change (1/1000) times the 2DTE. From these number, the 2DTE is 5.00791024 Hartree.

Alternatively, we can start from the reduced gradients. The value of the reduced gradient with respect to a displacement of the Al atom along the first reduced axis is 0.005007930232 (Hartree). At first order, this quantity is the product of the 2DTE by the reduced coordinate difference. The estimate of the 2DTE is thus 5.007930232 Hartree. The agreement with the other estimate is rather good (2.10^{-5} Hartree).

However, it is possible to do much better, thanks to the use of a higher–order finite–difference formula. For this purpose, one can perform another calculation, in which the change of reduced coordinate along the first axis is 0.002, instead of 0.001. The doubling of the perturbation allows for a rather simple higher–order estimation, as we will see later. The results of this calculation are as follows:

```
rms dE/dt=  7.1249E-03; max dE/dt=  1.0016E-02; dE/dt below (all hartree)
   1       0.010016299779      0.005097509981      0.005097509981
   2      -0.010016176675     -0.005097455174     -0.005097455174
...
Ewald  =-8.47987214716789E+00 resulting in Etotal=-9.76267372899697E+00 hartree
```

From these data, taking into account that the perturbation was twice stronger, the same procedure than above leads to the values 5.00800270 Hartree (from finite difference of energy) and 5.009149889 Hartree (from finite difference of forces, the value 0.010016299779 has to be multiplied by 1000/2). The combination of these data with the previous estimate can be done thanks to an higher–order finite–difference formula, in which the difference of estimations (the largest perturbation minus the smallest one) is divided by three, and then subtracted from the smallest estimation. As far as the total–energy estimation is concerned, the difference is 0.0001104 Ha, which divided by three, and subtracted from 5.00789230 Hartree, gives 5.0078555 Hartree. The same higher–order procedure for force estimates gives 5.0078552 Hartree. So, the agreement between total–energy estimate and force estimate of the 2DTE can be observed up to the 7th digit, inclusive.

Before comparing this result with the 2DTE directly computed from the response–function capabilities of ABINIT, a last comment is in order. One can observe that the action–reaction law is fulfilled only approximately by the system. Indeed, the force created on the second atom, should be exactly equal in magnitude to the force on the first atom. The values of dE/dt, mentioned above, for example for the doubled displacement:

```
rms dE/dt=  7.1249E-03; max dE/dt=  1.0016E-02; dE/dt below (all hartree)
   1       0.010016299779      0.005097509981      0.005097509981
   2      -0.010016176675     -0.005097455174     -0.005097455174
```

show a small, but non–negligible difference between the two atoms. Actually, the forces should cancel each other exactly if the translation symmetry were perfect. This is not the case, but the breaking of this symmetry can be shown to arise **only** from the presence of the exchange–correlation grid of points. This grid does not move when atoms are displaced, and so there is a very small variation of the total energy when the system is moved as a whole. It is easy to restore the action–reaction law, by subtracting from every force component the mean of the forces on all atoms. This is actually done when the gradient with respect to reduced coordinates are transformed into forces, and specified in Cartesian coordinates, as can be seen in the output file for the small displacement:

```
cartesian forces (hartree/bohr) at end:
   1     -0.00000421123276    -0.00047199792687    -0.00047199792687
   2      0.00000421123276     0.00047199792687     0.00047199792687
```

This effect will be seen also at the level of 2DTE. The so–called "acoustic sum rule", imposing that the frequency of three modes (called acoustic modes) tend to zero with vanishing wavevector, will also be slightly broken. In this case also, it will be rather easy to reimpose the acoustic sum rule. In any case, taking a finer XC grid will allow to reduce this effect.

5.5.3 Response–function calculation of a second derivative of the total energy

We now compute the second derivative of the total energy with respect to the same atomic displacement, using the response–function capabilities of ABINIT.

You can copy the file `~ABINIT/Tutorial/t53.in` in Work5. This is your input file. You should edit it. The changes with respect to the file `~ABINIT/Tutorial/t51.in` are all gathered in the first part of this file, before

```
############################################################################
#Common input variables
```

Accordingly, you should get familiarized with the new input variables: *rfphon*, *rfatpol*, *rfdir*. Then, pay attention to the special use of the *kptopt* input variable. It will be explained in more detail later.

When you have understood the purpose of the input variable values specified before the "Common input variables" section, you can make the code run. It takes less than one minute on a PIII 450MHz.

Then, we need to analyze the different output files. For that purpose, you should read the content of the section 6 of the `respfn_help` file. Read it quickly, as we will come back to the most important points hereafter.

ABINIT has created four different files:

- `t53.log` (the log file);
- `t53.out` (the output file);
- `t53o_1WF1` (the 1st–order wavefunction file);
- `t53o_DDB` (the derivative database).

Let us have a look at the output file. You can follow the description provided in the section 6.2 of the `respfn_help` file. You should be able to find the place where the iterations for the minimization (with respect to the unique perturbation) take place:

```
     iter   2DEtotal(Ha)       deltaE(Ha) residm    vres2
ETOT  1   6.5156051312863    -1.464E+01 1.146E-02 1.947E+02
ETOT  2   5.0216331638978    -1.494E+00 9.267E-04 2.027E+00
ETOT  3   5.0082678671217    -1.337E-02 4.772E-06 7.929E-02
ETOT  4   5.0078677958105    -4.001E-04 1.980E-07 2.712E-03
ETOT  5   5.0078558860285    -1.191E-05 6.103E-09 5.074E-05
ETOT  6   5.0078557520180    -1.340E-07 1.258E-10 9.613E-06
ETOT  7   5.0078557017091    -5.031E-08 2.768E-11 3.841E-07
ETOT  8   5.0078557001188    -1.590E-09 2.983E-12 6.089E-09
```

From these data, you can see that the 2DTE determined by the response–function technique is in excellent agreement with the higher–order finite–difference formula for the 2DTE, determined in the previous section: 5.0078555 Hartree from the energy differences, and 5.0078552 Hartree from the force differences.

Now, you can read the remaining of the section 6.2 of the `respfn_help` file. Then, you should also edit the `t53o_DDB` file, and read the corresponding section 6.4 of the `respfn_help` file.

Finally, the excellent agreement between the finite–difference formula and the response–function approach calls for some accuracy considerations. These can be found in section 7 of the `respfn_help` file.

5.5.4 Response–function calculation of the dynamical matrix at Gamma

We are now in the position to compute the full dynamical matrix at Gamma ($q = 0$). You can copy the file `~ABINIT/Tutorial/t54.in` in Work5. This is your input file. You should edit it. As for test 53, the changes with respect to the file `~ABINIT/Tutorial/t51.in` are all gathered in the first part of this file. Moreover, the changes with respect to `t53.in` concern only the input variables *rfatpol*, and *rfdir*. Namely, all the atoms will be displaced, in all the directions.

There are six perturbations to consider. So, one might think that the CPU time will raise accordingly. This is not true, as ABINIT is able to determine which perturbations are the symmetric of another perturbation, see section section 3 of the `respfn_help` file.

Now, you can make the run. It is a bit longer than one minute on a PIII at 450MHz. You edit the file `t54.out`, and notice that the response to two perturbations were computed explicitly, while the response to the other four could be deduced by using the symmetries. Nothing mysterious: one of the two irreducible perturbations is for the Al atom, placed in a rather symmetric local site, and the other perturbation is for the As atom.

The phonon frequencies, obtained by diagonalizing the dynamical matrix (where the atomic masses have been taken into account, see *amu*), are given as follows:

```
  Phonon wavevector (reduced coordinates) :  0.00000  0.00000  0.00000
 Phonon energies in Hartree :
   2.586632E-06  2.590723E-06  2.614440E-06  1.568560E-03  1.568560E-03
   1.568560E-03
 Phonon frequencies in cm-1    :
-  5.677000E-01  5.685980E-01  5.738033E-01  3.442590E+02  3.442590E+02
-  3.442590E+02
```

You might wonder about the dash sign present in the first column of the two lines giving the frequencies in cm−1. The first column of the main ABINIT output files is always dedicated to signs needed to automatic treat the comparison with respect to reference files. Except if you become a ABINIT developer, you should ignore these signs. In the present case, they should not be interpreted as a minus sign for the floating numbers that follow them.

There is a good news about this result, and a bad news. The good news is that there are indeed three acoustic mode, with frequency rather close to zero (less than 1 cm^{-1}, which is rather good!). The bad news comes when the three other frequencies are compared with experimental results, or other theoretical results. Indeed, in the present run, one obtains three degenerate modes, while there should be a (2+1) splitting. This can be seen in the paper "*Ab initio* calculation of phonon dispersions in semiconductors", by P. Giannozzi, S. de Gironcoli, P. Pavone and S. Baroni, Phys. Rev. B **43**, 7231 (1991) , especially Fig. 2.

Actually, we have forgotten to take into account the coupling between atomic displacements and the homogeneous electric field, that exists in the case of polar insulators, for so–called "Longitudinal Optic (LO) modes". A splitting appears between these modes and the "Transverse Optic (TO) modes". This splitting (Lyddane–Sachs–Teller LO–TO splitting) is presented in simple terms in standard textbooks, and should not be forgotten in doing *ab initio* calculations of phonon frequencies.

Thus we have now to treat correctly the homogeneous electric field type perturbation.

5.5.5 Response–function calculation of the effect of an homogeneous electric field

The treatment of the homogeneous electric field perturbation is formally much more complex than the treatment of atomic displacements. This is primarily because the change of potential associated with an homogeneous electric field is not periodic, and thus does not satisfy the Born–von Karman periodic boundary conditions.

For the purpose of the present tutorial, one should read the section II.C of the above–mentioned paper P. Giannozzi, S. de Gironcoli, P. Pavone and S. Baroni, Phys. Rev. B 43, 7231 (1991) .

The reader will find in X. Gonze, Phys. Rev. B 55, 10337 (1997) and X. Gonze and C. Lee, Phys. Rev. B 55, 10355 (1997) more detailed information of this perturbation, closely related to the ABINIT implementation. There is also an extensive discussion of the Born effective charges by Ph. Ghosez, J.-P. Michenaud and X. Gonze, Phys. Rev. B 58, 6224 (1998).

In order to compute the response of solids to an homogeneous electric field, as implemented in ABINIT, the remaining sections of the `respfn_help` file should be read. These also present the information needed to compute phonons with non–zero q–wavevector, which will be the subject of the next section of the present tutorial. The sections to be read are:

- the part of section 1 that had not yet been read;
- section 2;
- section 4;
- and, for completeness,section 5.

You are now in the position to compute the full dynamical matrix at Gamma ($q = 0$), including the coupling with an homogeneous electric field. You can copy the file `~ABINIT/Tutorial/t55.in` in Work5. This is your input file. You should edit it. As for the other RF tests, the changes with respect to the file `~ABINIT/Tutorial/t51.in` are all gathered in the first part of this file. Unlike the other tests, however, the multi–dataset mode was used, computing from scratch the ground–state properties, then computing the effect of the ddk perturbation, then the effect of all other perturbations (electric field as well as atomic displacements). The run lasts about 3 minutes on a PIII 450MHz.

The analysis of the output file is even more cumbersome than the previous ones. Let us skip the first dataset. In the dataset 2 section, one perturbation is correctly selected:

```
==>  initialize data related to q vector <==

The list of irreducible perturbations for this q vector is:
   1)    idir= 1    ipert=   3

================================================================================

--------------------------------------------------------------------------------
Perturbation wavevector (in red.coord.)   0.000000  0.000000  0.000000
Perturbation : derivative vs k along direction   1
```

The analysis of the output for this particular perturbation is not particularly interesting, except for the f–sum rule ratio

```
loper3 : ek2=    1.6833336546E+01
         f-sum rule ratio=    1.0028582985E+00
```

that should be close to 1, and becomes closer to it when *ecut* is increased, and the sampling of k–points is improved. (In the present status of ABINIT, the f–rule ratio is not computed correctly when *ecutsm*$\neq 0$)

In the third dataset section, three irreducible perturbations are considered:

```
==>  initialize data related to q vector <==

The list of irreducible perturbations for this q vector is:
   1)    idir= 1    ipert=   1
   2)    idir= 1    ipert=   2
   3)    idir= 1    ipert=   4
```

Much later, the dielectric tensor is given:

```
Dielectric tensor, in cartesian coordinates,
   j1       j2             matrix element
dir pert dir pert     real part    imaginary part

 1    4   1    4    9.7606048428    0.0000000000
 1    4   2    4    0.0000000000    0.0000000000
 1    4   3    4    0.0000000000    0.0000000000

 2    4   1    4    0.0000000000    0.0000000000
 2    4   2    4    9.7606048428    0.0000000000
 2    4   3    4    0.0000000000    0.0000000000

 3    4   1    4    0.0000000000    0.0000000000
 3    4   2    4    0.0000000000    0.0000000000
 3    4   3    4    9.7606048428    0.0000000000
```

It is diagonal and isotropic, and corresponds to a dielectric constant of 9.7606048428.

Then, the Born effective charges are given, either computed from the derivative of the wavefunctions with respect to the electric field, or computed from the derivative of the wavefunctions with respect to an atomic displacement, as explained in section II of X. Gonze, Phys. Rev. B 55, 10355 (1997):

```
Effective charges, in cartesian coordinates,
(from electric field response)
...
```

and

```
Effective charges, in cartesian coordinates,
(from phonon response)
...
```

Namely, the Born effective charge of the Al atom is 2.104, and the one of the As atom is −2.127. The charge neutrality sum rule is not fulfilled exactly. When *ecut* is increased, and the sampling of k–points is improved, the sum of the two charges goes closer to zero.

Finally, the phonon frequencies are computed:

```
  Phonon wavevector (reduced coordinates) :  0.00000  0.00000  0.00000
 Phonon energies in Hartree :
   2.586632E-06  2.590723E-06  2.614440E-06  1.568560E-03  1.568560E-03
   1.568560E-03
 Phonon frequencies in cm-1    :
-  5.677000E-01  5.685980E-01  5.738033E-01  3.442590E+02  3.442590E+02
-  3.442590E+02

  Phonon at Gamma, with non-analyticity in the
  direction (cartesian coordinates)  1.00000  0.00000  0.00000
 Phonon energies in Hartree :
   2.590670E-06  2.590723E-06  4.101029E-06  1.568560E-03  1.568560E-03
   1.729575E-03
 Phonon frequencies in cm-1    :
-  5.685864E-01  5.685980E-01  9.000719E-01  3.442590E+02  3.442590E+02
-  3.795979E+02

  Phonon at Gamma, with non-analyticity in the
  direction (cartesian coordinates)  0.00000  1.00000  0.00000
```

```
 Phonon energies in Hartree :
   2.586632E-06  2.614440E-06  4.088526E-06  1.568560E-03  1.568560E-03
   1.729575E-03
 Phonon frequencies in cm-1    :
-  5.677000E-01  5.738033E-01  8.973277E-01  3.442590E+02  3.442590E+02
-  3.795979E+02

  Phonon at Gamma, with non-analyticity in the
  direction (cartesian coordinates)  0.00000  0.00000  1.00000
 Phonon energies in Hartree :
   2.590723E-06  2.610339E-06  4.088540E-06  1.568560E-03  1.568560E-03
   1.729575E-03
 Phonon frequencies in cm-1    :
-  5.685980E-01  5.729031E-01  8.973308E-01  3.442590E+02  3.442590E+02
-  3.795979E+02
```

The first few lines discard any effect of the homogeneous electric field, while the next sections consider it along the three Cartesian coordinates.

In the present material, the directionality of the electric field has no influence. We note that there are still three acoustic mode, below 1 cm^{-1}, while the optic modes have the correct degeneracies: two TO modes at 344.3 cm^{-1}, and one LO mode at 379.6 cm^{-1}.

These values can be compared to experimental (361 cm^{-1}, 402 cm^{-1}) as well as theoretical (363 cm^{-1}, 400 cm^{-1}) values (again the Gianozzi et al paper mentioned above). Most of the discrepancy comes from the too low value of *ecut*. Using ABINIT with *ecut*=6 Hartree gives (358.8 cm^{-1}, 389.8 cm^{-1}). The remaining of the discrepancy may come partly from the pseudopotentials, that are particularly soft.

The comparison of Born effective charges is also interesting. After imposition of the neutrality sum rule, the Al Born effective charge is 2.116. The value from Gianozzi *et al* is 2.17, the experimental value is 2.18. Increasing *ecut* to 6 Hartree in ABINIT gives 2.168.

For the dielectric tensor, it is more delicate. The value from Gianozzi et al is 9.2, while the experimental value is 8.2. The agreement is not very good, a fact that can be attributed to the LDA lack of polarization–dependence (X. Gonze, Ph. Ghosez and R. Godby, Phys. Rev. Lett. (1995)). Still, the agreement of our calculation with the theoretical result is not very good. With $ecut = 3$ Hartree, we have 9.76. Changing it to 6 Hartree gives 10.40. A better k–point sampling (8x8x8), with $ecut = 6$ Hartree, reduces the value to 9.89. Changing pseudopotentials finally improves the agreement: with the much harder `13al.pspgth` and `33as.psphgh` pseudopotentials with adequate $ecut = 16$ Hartree and 8x8x8 Monkhorst–Pack sampling, we reach a value of 9.37. This illustrates that the dielectric tensor is a much more sensitive quantity than the others.

5.5.6 Response–function calculation of phonon frequencies at non–zero q

The computation of phonon frequencies at non–zero q is actually simpler than the one at Gamma. One must distinguish two cases. Either the q–wavevector connects k–points that belong to the same grid, or the wavevector q is general. In any case, the computation within the response–function formalism is more efficient than using the frozen–phonon technique: the use of supercell is completely avoided. For an explanation of this fact, see for example section IV of X. Gonze, Phys. Rev. B55, 10337 (1997).

You can copy the file `~ABINIT/Tutorial/t56.in` in Work5. This is your input file. You should edit it. As for the other RF tests, the changes with respect to the file `~ABINIT/Tutorial/t51.in` are all gathered in the first part of this file. The multi–dataset mode is used, computing from scratch the ground–state properties, then computing different dynamical matrices. The run is rather long: about 30 minutes on a PIII 450MHz. So, you would better leave your computer

running, and either read more of the ABINIT documentation (why not the `mrgddb_help` and the `anaddb_help`), or make a walk.

The results of this simulation can be compared to those provided in the Gianozzi et al paper. The agreement is rather good, despite the low cut–off energy, and different pseudopotentials.

At X, they get 95 cm^{-1}, 216 cm^{-1}, 337 cm^{-1} and 393 cm^{-1}, while we get 92.5 cm^{-1}, 204.6 cm^{-1}, 313.9 cm^{-1} and 375.9 cm^{-1}. With *ecut* = 6 Hartree, we get 89.7 cm^{-1}, 212.3 cm^{-1}, 328.5 cm^{-1} and 385.8 cm^{-1}.

At L, they get 71 cm^{-1}, 212 cm^{-1}, 352 cm^{-1} and 372 cm^{-1}, while we get 69.0 cm^{-1}, 202.5 cm^{-1}, 332.6 cm^{-1} and 352.3 cm^{-1}. With *ecut* = 6 Hartree, we get 68.1 cm^{-1}, 208.5 cm^{-1}, 346.7 cm^{-1} and 362.6 cm^{-1}.

At q=(0.1 0 0), we get 31.6 cm^{-1}, 63.6 cm^{-1}, 342.0 cm^{-1} and 379.7 cm^{-1}. The acoustic modes tends (nearly-)linearly to zero, while the optic modes are close to their values at Gamma: 344.3 cm^{-1} and 379.6 cm^{-1}.

5.5.7 The computation of full phonon band structures and thermodynamical properties

This section is still to be written. You might have a look at the tests 26 to 32 of the directory `~ABINIT/Infos/Test_v2`.

The ABINIT tutorial is now finished. It might be worth to read the full list of abinis input variables. Then, proceeds to the `~ABINIT/Infos/Dirs_and_files` file, to have a global view of ABINIT.

5.6 Lesson 6: The quasi–particle band structure of Silicon, in the GW approximation

This lesson aims at showing how to get self–energy corrections to the DFT Kohn–Sham eigenvalues in the GW approximation. The GW formalism will NOT be explained in this tutorial. The reader might consult the review

- "Quasiparticle calculations in solids", by Aulbur W G, Jonsson L, Wilkins J W, in Solid State Physics 54, 1–218 (2000), also available at http://www.physics.ohio-state.edu/~wilkins/vita/gw_review.ps

The different formulas of the GW formalism, that have been implemented in ABINIT, have been written in a pdf document by Valerio Olevano (who also wrote the first version of this tutorial), see `~ABINIT/Infos/Theory/gwa.pdf`.

This lesson should take about 2 hours to be done.

5.6.1 Computation of the Silicon band gap at Gamma, using a GW calculation

Before beginning, you might consider to work in a different subdirectory as for the other lessons. Why not "Work6"?

At the end of lesson 3, you computed the Kohn–Sham band structure of Silicon. In this approximation, the variation of eigenvalues inside each band is reasonable, as well as the band widths, but the band gaps are known to be qualitatively wrong. Now, we will compute the band gaps much more accurately, using the so–called GW approximation.

We start by an example, in which we show how to perform in one shot the calculation of the ground state, the Kohn Sham electronic structure, the screening, and the Self–Energy matrix elements, that is, the GW corrections, for one given k–point, for the highest occupied and the lowest empty bands. We provide some reasonable parameters without checking convergence. You

will see that this procedure is MUCH MORE time–consuming than the corresponding calculation of the Kohn–Sham eigenvalues.

So, let us run immediately this calculation, and while it is running, we will explain what has been done.

In directory `~ABINIT/Tutorial/Work6`, copy the files `~ABINIT/Tutorial/t6x.files` and `t61.in`, and modify the `t6x.files` file as usual. Then, issue:

```
../../abinis < t6x.files >& t61.log &
```

It is very important to run this job in background. Indeed, a PC Intel PIV/2.2 GHz will take about 6 minutes to complete it. In the meantime, you should read the following.

1. **The three steps of a GW calculation.**

   In order to complete a standard GW calculation, one has to:

   (a) Run a converged Ground State calculation (at fixed lattice parameters and atomic positions), to get self–consistent density and potential, and Kohn–Sham eigenvalues and eigenfunctions at the relevant k–point as well as on a regular grid of k–points;

   (b) On the basis of these available Kohn–Sham data, compute the independent–particle susceptibility matrix ("chi0"), on a regular grid of wavevectors, for at least two frequencies (usually, zero frequency and a large frequency — on the order of the plasmon frequency, a dozen of eV), then compute the dielectric matrix ("epsilon") in the same conditions, its inverse, and the Random–Phase susceptibility matrix ("chi");

   (c) On this basis, compute the self–energy operator ("sigma") at a given k–point, and derive the GW eigenvalues for the target states at this k–point.

   The input file `t61.in` has precisely that structure: there are three datasets. The first dataset starts a rather usual SCF calculation, then will construct a specialized file, `t6xo_DS1_KSS` (`_KSS` for "Kohn–Sham Structure"), that contains the needed information to start step 2. The second dataset drives the computation of susceptibility and dielectric matrices, giving another specialized file, `t6xo_DS2_EM1` (`_EM1` for "Epsilon Minus 1" — the inverse dielectric matrix). Then, in the third dataset, one builds the eigenvalues of the 4th and 5th bands at the Gamma point.

   So, you can edit this `t61.in` file.

   In the dataset–independent part of this file (the last half of the file), there is the usual set of input variables, describing the cell, atom types, number, position, planewave cut–off energy, SCF convergence parameters, than in the `t35.in` file, driving the Kohn–Sham band structure calculation. Then, for the three datasets, you will find specialized additional input variables.

2. **Generating the Kohn–Sham band structure: the KSS file.**

   In dataset 1, apart from the usual input variables we are acquainted to through the previous tutorials, there is a new input variable:

   ```
   nbandkss -1       # Number of bands in KSS file (-1 means the maximum possible)
   ```

   This input variable tells the program to calculate the Kohn–Sham electronic structure by the (in this case) full diagonalization of the Kohn–Sham Hamiltonian evaluated at the converged density and calculated in each one of the k–points of the grid. Note that this diagonalization is performed in a routine (`outkss.f`) separated from the usual SCF cycle, so that there is additional control of the wavefunction actually stored, if needed. In particular, the number of bands to be computed in this routine is NOT determined by the usual input variable *nband*.

nbandkss is the key variable to create a _KSS file. If it is zero, no _KSS file will be created. −1 lead to the generation and storage of the maximum possible number of states (or bands) common to all points. This might lead to quite time–consuming calculations. One can reduce the load in the diagonalization by requiring less states.

Another way to reduce the load of the diagonalization is made possible through the use of *npwkss*. It governs the size of the plane wave basis set in which the Hamiltonian matrix will be expressed and diagonalized. The default value leaves the number of plane wave equal to the one of the SCF ground state calculation.

Another relevant input variable, related also to the specific set up of the _KSS file is *kssform*.

In this first dataset, we asked also the self–consistent cycle to be done for nine bands.

```
nband1      9          # Number of (occ and empty) bands to be computed
```

Only four bands would be needed for Si. The purpose of defining more bands in the ground–state determination is to verify that at least the first Kohn–Sham eigenvalues obtained through the diagonalization are sufficiently close to those determined in the self–consistent procedure. At present, the comparison is not done automatically, so please check (well, sometimes ...) that the Kohn–Sham eigenvalues given in the self–consistency part (with a residual) are close to those given after the diagonalization.

3. **Generating the screening: the EM1 file.**

In dataset 2 the calculation of the screening (susceptibility, dielectric matrix) is performed. We need to set *optdriver*=3 to do that:

```
optdriver2  3         # Screening calculation
```

The *getkss* input variable is similar to other "get" input variables of ABINIT:

```
getkss2     -1        # Obtain KSS file from previous dataset
```

In this case, it tells the code to use the KSS file calculated in the previous dataset.

Then, three input variables describe the computation:

```
nband2      25        # Bands to be used in the screening calculation
ecutwfn2    2.1       # Cut--off energy of the planewave set to
                      # represent the wavefunctions
ecuteps2    3.6       # Cut--off energy of the planewave set to
                      # represent the dielectric matrix
```

In this case, we use 25 bands to calculate the Kohn–Sham response function $\chi^{(0)}_{KS}$; we use a cut–off *ecutwfn*=2.1 Hartree, giving 89 planewaves to represent the wavefunctions in the calculation of $\chi^{(0)}_{KS}$. The dimension of $\chi^{(0)}_{KS}$, as well as all the other matrices (χ, ϵ) is determined by *ecuteps*=3.6 Hartree, giving 169 planewaves.

Finally we define the frequencies at which the screening must be evaluated: $\omega = 0.0$ eV and the imaginary frequency $\omega =$ i 16.7 eV. The latter is determined by the input variable *plasfrq*:

```
plasfrq2    16.7 eV   # Imaginary frequency where to calculate the
                      # screening
```

The two frequencies are used to calculate the plasmon–pole model parameters. For the non–zero frequency it is recommended to use a value close to the plasmon frequency for the plasmon–pole model to work well. Plasmons frequencies are usually close to 0.5 Hartree. The parameters for the screening calculation are not far from the ones that give converged Energy Loss Function ($-\mathrm{Im}\ \epsilon_{00}^{-1}$) spectra, So that one can start up by using indications from EELS calculations existing in literature.

4. **Computing the GW energies.**

   In dataset 3 the calculation of the Self–Energy matrix elements is performed. One needs to define the driver option, as well as the `_KSS` and `_EM1` files.

```
optdriver3  4        # Self-Energy calculation
getkss3     -2       # Obtain KSS file from dataset 1
geteps3     -1       # Obtain EM1 file from previous dataset
```

   The *geteps* input variable is also similar to other "get" input variables of ABINIT.

   Then, comes the definition of parameters needed to compute the self–energy. As for the computation of the susceptibility and dielectric matrices, one must define the set of bands, and two sets of planewaves:

```
nband3      100      # Bands to be used in the Self-Energy calculation
ecutwfn3    5.0      # Planewaves to be used to represent the wave
                     # functions
ecutsigx3   6.0      # Dimension of the G sum in Sigma_x
                     # (the dimension in Sigma_c is controlled by npweps)
```

   In this case, *nband* controls the number of bands used to calculate the Self–Energy. *ecutwfn* defines (as for *optdriver* = 3) the number of planewaves used to represent the wavefunctions. *ecutmat* gives the dimension of the planewave sum needed to calculate Sigma_x (the exchange part of the self–energy, which is diagonal). The size of the planewave set needed to compute Sigma_c (the correlation part of the self–energy) is controlled by the dimension of the screening matrix read in the EM1 file. However, it is taken equal to the number of planewave of Sigma_x if the latter is smaller than the one for Sigma_c.

   Then, come the parameters defining the k–points and states for which the electronic energy must be computed:

```
nkptgw3      1               # number of k-point where to calculate the
                             # GW correction
kptgw3                       # k-points
  -0.125    0.000    0.000
bdgw3       4  5             # calculate GW corrections for bands from 4 to 5
```

   nkptgw tells the number of k–points for which the GW corrections must be computed. The k–points coordinates are given in *kptgw*. At present, they must belong to the grid of k–points defined with the same repetition parameters (*kptrlatt*, or *ngkpt*) than the GS one, but WITHOUT any shift. *bdgw* gives the minimum/maximum band whose energies are calculated.

   There is an additional parameter, called *zcut*, related to the self–energy computation. It is meant to avoid some divergences that might occur in the calculation due to integrable poles along the integration path.

5. **Examination of the output file.**

   Let us hope now that your calculation has been completed, and that we can examine the output file. So, please edit the `t61.out` file.

The first departure from the usual information present in the output file for usual GS calculations appears after the SCF cycles of DATASET 1:

```
================================================================
 Calculating and writing out Kohn-Sham electronic Structure file
 Using diagonalized wavefunctions and energies (kssform=1)
 number of Gamma centered plane waves    483
 number of Gamma centered shells     25
 number of bands    283
```

This section was issued when the Hamiltonian at the different k–points were diagonalized, after the SCF cycles, in order to generate the KSS file. Then, comes the output of the numerous eigenvalues at the different k–points. Finally, the normalization and orthogonalization of the eigenvectors is tested. One should obtain perfect normalization and orthogonalization at that stage:

```
Test on the normalization of the wavefunctions
 min sum_G |a(n,k,G)| =  1.000000
 max sum_G |a(n,k,G)| =  1.000000
Test on the orthogonalization of the wavefunctions
 min sum_G a(n,k,G)* a(n',k,G) =  0.000000
 max sum_G a(n,k,G)* a(n',k,G) =  0.000000
```

Then, follows the usual information for the dataset 1. The dataset 2 drives the computation of the susceptibility and dielectric matrices, in preparation of the GW energy calculation of dataset 3. After some general information (origin of KSS file, header, description of unit cell), comes the echo of Kohn–Sham eigenenergies (in eV), and then the evaluation of the wavefunction normalization and orthogonalization USING ONLY THE PLANEWAVE SET DEFINED BY *ecutwfn*, *npwwfn*, or *nshwfn*. Thus, there is no surprise that these relations are not fulfilled:

```
test on the normalization of the wavefunctions
min sum_G |a(n,k,G)| =  0.497559
max sum_G |a(n,k,G)| =  0.995840
test on the orthogonalization of the wavefunctions
min sum_G a(n,k,G)* a(n",k,G) =  0.000000
max sum_G a(n,k,G)* a(n",k,G) =  0.179460
```

The squared norm of one of the wavefunctions is even as low as one half! This should lead us to question the choice of *ecutwfn* that we have made: we will need a convergence study, see later.

The parameters of the FFT grid needed to represent the wavefunction and compute their convolution (so as to get the screening matrices) are then given.

Then, the grid of q–point (in the irreducible part of the Brillouin Zone) on which the susceptibility and dielectric matrices will be computed is given. It is a grid of points with the same repetition parameters (*kptrlatt*, or *ngkpt*) than the GS one, but WITHOUT any shift.

On the basis of only the average density, one can obtain the plasmon frequency of metallic Jellium (homogeneous electron gas, placed in a neutralizing background). The next lines start from the average density of the system, and evaluate the r_s parameter of the Jellium, then compute the plasmon frequency. THIS IS A ROUGH ESTIMATE. In particular, it will be questionable for strongly inhomogeneous systems! Also, the choice of pseudopotential (inclusion of core states) will have an effect on this estimate. So, take it cautiously. It is better to try a few values of *plasfrq* than to rely blindly on this value ...

At the end of the screening calculation, the macroscopic dielectric constant is printed:

```
dielectric constant = 13.8476
dielectric constant without local fields = 15.5520
```

Note that the convergence in the dielectric constant DOES NOT guarantee the convergence in the GW correction values at the end of the calculation. In fact, the dielectric constant is representative of only one element, the head, of the full dielectric matrix. Even if the convergence on the dielectric constant with local fields takes somehow into account also other non–diagonal elements. In a GW calculation all the ϵ^{-1} matrix is used to build the Self–Energy operator.

The dielectric constant here reported is the so–called RPA dielectric constant due to the electrons. Although evaluated at zero frequency, it is understood that the ionic response is not included. This is to be contrasted with the one computed in ANADDB). The RPA dielectric constant restricted to electronic effects is also not the same as the one computed in the RESPFN part of ABINIT, that includes exchange–correlation effects.

We enter now the third dataset. As for dataset 2, after some general information (origin of KSS file, header, description of unit cell), the echo of Kohn–Sham eigenenergies (in eV), the evaluation of the wavefunction normalization, the description of the FFT grid and Jellium parameters, there is the echo of parameters for the plasmon–pole model, and the inverse dielectric function (the screening). The self–energy operator has been constructed, and one can evaluate the GW energies, for each of the states.

The results follows:

```
k =   -0.125   0.000   0.000
 Band     E0  <VxcLDA>   SigX SigC(E0)    Z dSigC/dE  Sig(E)    E-E0     E
    4   5.616 -11.132 -12.334   1.257   0.775  -0.290 -11.089   0.043   5.659
    5   8.357 -10.157  -5.951  -3.336   0.779  -0.284  -9.480   0.677   9.034

 E^0_gap           2.741
 E^GW_gap          3.375
 DeltaE^GW_gap     0.634
```

For the desired k–point, state 4, then state 5, one finds different information:

- `E0` is the Kohn–Sham eigenenergy;
- `VxcLDA` gives the average Kohn–Sham exchange–correlation potential;
- `SigX` gives the exchange contribution to the self–energy;
- `SigC(E0)` gives the correlation contribution to the self–energy, evaluated at the Kohn–Sham eigenenergy;
- `Z` is the renormalization factor;
- `dSigC/dE` is the energy derivative of `SigC` with respect to the energy;
- `SigC(E)` gives the correlation contribution to the self–energy, evaluated at the GW energy;
- `E-E0` is the difference between GW energy and Kohn–Sham eigenenergy;
- `E` is the GW energy.

In this case, the gap is also analyzed: `E^0_gap` is the Kohn–Sham one, `E^GW_gap` is the GW one, and `DeltaE^GW_gap` is the difference.

It is seen that the average Kohn–Sham exchange–correlation potential for the state 4 (a valence state) is very close to the exchange self–energy correction. For that state, the correlation correction is small, and the difference between Kohn–Sham and GW energies is also small (43 meV). By contrast, the exchange self–energy is much smaller than the average

Kohn–Sham potential for the state 5 (a conduction state), but the correlation correction is much larger than for state 4. On the whole, the difference between Kohn–Sham and GW energies is not very large, but nevertheless, it is quite important when compared with the size of the gap.

5.6.2 Preparing convergence studies: Kohn–Sham structure (KSS file) and screening (EM1 file)

In the following sections, we will perform different convergence analyses, because this is such an important task. In order to keep the CPU time at a reasonable level, we will use fake KSS and screening data, by limiting ourselves to the Gamma point only. In that way, we will verify convergence aspects that could be very cumbersome (at least in the framework of a tutorial) if more k–points were used.

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t62.in`, and modify the `t6x.files` file as usual. Edit the `t62.in` file, and take the time to examine it. Note that the SCF cycles have been disconnected from the generation of the KSS file. Then, issue:

```
../../abinis < t6x.files >& t62.log &
```

This small job lasts about 10 secs on a PC PIV Intel 2.2 GHz.

After that step you will need the KSS and EM1 files produced in this run for the next runs (up to 6.8). Move `t6xo_DS2_KSS` to `t6xo_DS1_KSS` and `t6xo_DS3_EM1` to `t6xo_DS1_EM1`.

The next 6 sections are intended to show you how to find the converged parameters for a GW calculation.

5.6.3 Convergence on the number of planewaves in the wavefunctions to calculate the Self–Energy

We begin by the convergence study on the three parameters needed in the self–energy calculation (*optdriver*=4). This is because for these, we will not need a double dataset loop to check this convergence, and we will rely on the previously determined EM1 file.

First, we check the convergence on the number of planewaves to describe the wavefunctions, in the calculation of the Self–Energy. This will be done by defining five datasets, with increasing *ecutwfn*:

```
ndtset     5
ecutwfn:  3.0
ecutwfn+  1.0
```

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t63.in`, and modify the `t6x.files` file as usual. Edit the `t63.in` file, and take the time to examine it. Then, issue:

```
../../abinis < t6x.files >& t63.log &
```

This small job lasts about 10 secs on a PC PIV Intel 2.2 GHz.

Edit the output file. The number of plane waves used for the wavefunctions in the computation of the self–energy is mentioned in the fragments of output:

```
SIGMA fundamental parameters:
PLASMON POLE MODEL
number of plane-waves for SigmaX                 169
number of plane-waves for SigmaC and W           169
number of plane-waves for wavefunctions           59
```

Gathering the GW energies for each planewave set, one gets:

```
number of plane-waves for wavefunctions           59
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.651 -15.237   3.897   0.806  -0.240 -11.401   0.251   6.166
   5   8.445  -9.669  -3.222  -5.460   0.819  -0.222  -8.861   0.808   9.253

number of plane-waves for wavefunctions          113
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.244   3.789   0.804  -0.244 -11.495   0.159   6.075
   5   8.445  -9.691  -3.213  -5.564   0.817  -0.224  -8.944   0.747   9.192

number of plane-waves for wavefunctions          137
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.244   3.779   0.804  -0.244 -11.502   0.151   6.066
   5   8.445  -9.702  -3.216  -5.577   0.817  -0.225  -8.960   0.743   9.188

number of plane-waves for wavefunctions          169
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.651 -15.242   3.770   0.804  -0.245 -11.508   0.144   6.059
   5   8.445  -9.718  -3.221  -5.584   0.817  -0.225  -8.972   0.745   9.190

number of plane-waves for wavefunctions          259
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.667 -15.253   3.766   0.803  -0.245 -11.522   0.145   6.060
   5   8.445  -9.716  -3.219  -5.591   0.816  -0.225  -8.977   0.740   9.185
```

So that *npwwfn*=137 (*ecutwfn*=5.0) can be considered converged within 0.01eV.

5.6.4 Convergence on the number of planewaves to calculate Sigma_x

Second, we check the convergence on the number of planewaves in the calculation of the Sigma_X. This will be done by defining five datasets, with increasing *ecutmat*:

```
ndtset     7
ecutsigx:  3.0
ecutsigx+  1.0
```

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t64.in`, and modify the `t6x.files` file as usual. Edit the `t64.in` file, and take the time to examine it. Then, issue:

```
../../abinis < t6x.files >& t64.log &
```

This small job lasts about 12 secs on a PC PIV Intel 2.2 GHz.

Edit the output file. The number of plane waves used for Sigma_X is mentioned in the fragments of output:

```
SIGMA fundamental parameters:
PLASMON POLE MODEL
number of plane-waves for SigmaX                 59
number of plane-waves for SigmaC and W           59
```

Gathering the GW energies for each planewave set, one gets:

```
number of plane-waves for SigmaX                 59
number of plane-waves for SigmaC and W           59
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.195   3.862   0.806  -0.241 -11.395   0.259   6.174
   5   8.445  -9.702  -3.177  -5.595   0.818  -0.223  -8.941   0.761   9.206
```

```
number of plane-waves for SigmaX                 113
number of plane-waves for SigmaC and W           113
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.235   3.795   0.804  -0.244 -11.482   0.172   6.087
   5   8.445  -9.702  -3.210  -5.581   0.817  -0.224  -8.958   0.744   9.189

number of plane-waves for SigmaX                 137
number of plane-waves for SigmaC and W           137
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.241   3.785   0.804  -0.244 -11.495   0.159   6.074
   5   8.445  -9.702  -3.213  -5.577   0.817  -0.224  -8.958   0.745   9.190

number of plane-waves for SigmaX                 169
number of plane-waves for SigmaC and W           169
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.244   3.779   0.804  -0.244 -11.502   0.151   6.066
   5   8.445  -9.702  -3.216  -5.577   0.817  -0.225  -8.960   0.743   9.188

number of plane-waves for SigmaX                 259
number of plane-waves for SigmaC and W           169

Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.247   3.779   0.804  -0.244 -11.504   0.150   6.065
   5   8.445  -9.702  -3.218  -5.577   0.817  -0.225  -8.961   0.741   9.186

number of plane-waves for SigmaX                 283
number of plane-waves for SigmaC and W           169
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.247   3.779   0.804  -0.244 -11.504   0.150   6.065
   5   8.445  -9.702  -3.218  -5.577   0.817  -0.225  -8.961   0.741   9.186

number of plane-waves for SigmaX                 283
number of plane-waves for SigmaC and W           169
Band     E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.654 -15.247   3.779   0.804  -0.244 -11.504   0.150   6.065
   5   8.445  -9.702  -3.218  -5.577   0.817  -0.225  -8.961   0.741   9.186
```

So that *npwsigx*=169 (*ecutsigx*=6.0) can be considered converged within 0.01 eV.

5.6.5 Convergence on the number of bands to calculate the Self–Energy

At last, as concerns the computation of the self–energy, we check the convergence on the number of bands in the calculation of the Sigma_X. This will be done by defining five datasets, with increasing *nband*:

```
ndtset  5
nband:  50
nband+  50
```

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t65.in`, and modify the `t6x.files` file as usual. Edit the `t65.in` file, and take the time to examine it. Then, issue:

```
../../abinis < t6x.files >& t65.log &
```

This small job lasts about 12 secs on a PC PIV Intel 2.2 GHz.

Edit the output file. The number of bands used for the self–energy is mentioned in the fragments of output:

```
SIGMA fundamental parameters:
PLASMON POLE MODEL
number of plane-waves for SigmaX                 169
number of plane-waves for SigmaC and W           169
number of plane-waves for wavefunctions          137
number of bands                                   50
```

Gathering the GW energies for each number of bands, one gets:

```
number of bands                                   50
   4   5.915 -11.654 -15.244   3.853   0.804  -0.243 -11.443   0.211   6.126
   5   8.445  -9.702  -3.216  -5.507   0.817  -0.224  -8.902   0.800   9.246

number of bands                                  100
   4   5.915 -11.654 -15.244   3.779   0.804  -0.244 -11.502   0.151   6.066
   5   8.445  -9.702  -3.216  -5.577   0.817  -0.225  -8.960   0.743   9.188

number of bands                                  150
   4   5.915 -11.654 -15.244   3.771   0.804  -0.244 -11.509   0.145   6.060
   5   8.445  -9.702  -3.216  -5.585   0.817  -0.225  -8.966   0.736   9.182

number of bands                                  200
   4   5.915 -11.654 -15.244   3.769   0.804  -0.244 -11.510   0.143   6.059
   5   8.445  -9.702  -3.216  -5.587   0.817  -0.225  -8.967   0.735   9.180

number of bands                                  250
   4   5.915 -11.654 -15.244   3.769   0.804  -0.244 -11.510   0.143   6.058
   5   8.445  -9.702  -3.216  -5.587   0.817  -0.225  -8.967   0.735   9.180
```

So that *nband*=100 can be considered converged within 0.01 eV.

At this stage, we know that for the self–energy computation, we need *ecutwfn*=5.0 *ecutmat*=6.0, *nband*=100.

5.6.6 Convergence on the number of planewaves in the wavefunctions to calculate the screening (ϵ^{-1})

Now, we come back to the calculation of the screening. Adequate convergence studies will couple the change of parameters for *optdriver*=3 with a computation of the GW energy changes. One cannot rely on the convergence of the macroscopic dielectric constant to assess the convergence of the GW energies.

As a consequence, we will define a double loop over the datasets:

```
ndtset      10
udtset      5  2
```

The datasets 12, 22, 32, 42 and 52, drive the computation of the GW energies:

```
# Calculation of the Self--Energy matrix elements (GW corrections)
optdriver?2  4
geteps?2     -1
ecutwfn?2    5.0
ecutsigx      6.0
nband?2      100
```

The datasets 11,21,31,41 and 51, drive the corresponding computation of the screening:

```
# Calculation of the screening (epsilon^-1 matrix)
optdriver?1  3
```

In this latter series, we will have to vary the three different parameters *ecutwfn*, *ecuteps* and *nband*.

First, we check the convergence on the number of planewaves to describe the wavefunctions, in the calculation of the screening. This will be done by defining five datasets, with increasing *ecutwfn*:

```
ecutwfn:?   3.0
ecutwfn+?   1.0
```

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t66.in`, and modify the `t6x.files` file as usual. Edit the `t66.in` file, and take the time to examine it. Then, issue:

```
../../abinis < t6x.files >& t66.log &
```

This small job lasts about 15 secs on a PC PIV Intel 2.2 GHz.

Edit the output file. The number of plane waves used for the wavefunctions in the computation of the screening is mentioned in the fragments of output:

```
EPSILON^-1 parameters (EM1 file):
dimension of the eps^-1 matrix                   169
number of plane-waves for wavefunctions           59
```

Gathering the macroscopic dielectric constant and GW energies for each planewave set, one gets:

```
dielectric constant = 101.5301
dielectric constant without local fields = 147.3095
number of plane-waves for wavefunctions            59
   4   5.915 -11.654 -15.244   3.799   0.806  -0.241 -11.486   0.168   6.083
   5   8.445  -9.702  -3.216  -5.555   0.816  -0.225  -8.942   0.761   9.206

dielectric constant =  99.5265
dielectric constant without local fields = 143.7208
number of plane-waves for wavefunctions           113
   4   5.915 -11.654 -15.244   3.769   0.804  -0.244 -11.510   0.143   6.059
   5   8.445  -9.702  -3.216  -5.582   0.815  -0.226  -8.964   0.738   9.183

dielectric constant =  98.2598
dielectric constant without local fields = 142.5982
number of plane-waves for wavefunctions           137
   4   5.915 -11.654 -15.244   3.762   0.801  -0.248 -11.517   0.137   6.052
   5   8.445  -9.702  -3.216  -5.588   0.815  -0.227  -8.970   0.733   9.178

dielectric constant =  97.6265
dielectric constant without local fields = 142.1664
number of plane-waves for wavefunctions           169
   4   5.915 -11.654 -15.244   3.759   0.804  -0.244 -11.519   0.135   6.050
   5   8.445  -9.702  -3.216  -5.590   0.815  -0.227  -8.972   0.731   9.176

dielectric constant =  96.4286
dielectric constant without local fields = 140.5466
number of plane-waves for wavefunctions           259
   4   5.915 -11.654 -15.244   3.760   0.803  -0.245 -11.518   0.136   6.051
   5   8.445  -9.702  -3.216  -5.592   0.815  -0.227  -8.973   0.730   9.175
```

So that *npwwfn*=113 (*ecutwfn*=4.0) can be considered converged within 0.01 eV.

5.6.7 Convergence on the number of bands to calculate the screening

Second, we check the convergence on the number of bands in the calculation of the screening. This will be done by defining five datasets, with increasing *nband*:

```
nband11  25
nband21  50
nband31  100
nband41  150
nband51  200
```

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t67.in`, and modify the `t6x.files` file as usual. Edit the `t67.in` file, and take the time to examine it. Then, issue:

```
../../abinis < t6x.files >& t67.log &
```

This small job lasts about 22 secs on a PC PIV Intel 2.2 GHz.

Edit the output file. The number of bands used for the wavefunctions in the computation of the screening is mentioned in the fragments of output:

```
EPSILON^-1 parameters (EM1 file):
dimension of the eps^-1 matrix                    169
number of plane-waves for wavefunctions           113
number of bands                                    25
```

Gathering the macroscopic dielectric constant and GW energies for each number of bands, one gets:

```
dielectric constant =  99.5265
dielectric constant without local fields = 143.7208
number of bands                                    25
   4   5.915 -11.654 -15.244   3.769   0.804  -0.244 -11.510   0.143   6.059
   5   8.445  -9.702  -3.216  -5.582   0.815  -0.226  -8.964   0.738   9.183

dielectric constant = 100.6436
dielectric constant without local fields = 143.7240
number of bands                                    50
   4   5.915 -11.654 -15.244   3.587   0.804  -0.244 -11.657  -0.003   5.912
   5   8.445  -9.702  -3.216  -5.764   0.815  -0.227  -9.113   0.589   9.034

dielectric constant = 101.1764
dielectric constant without local fields = 143.7244
number of bands                                   100
   4   5.915 -11.654 -15.244   3.516   0.804  -0.244 -11.714  -0.060   5.855
   5   8.445  -9.702  -3.216  -5.846   0.811  -0.233  -9.182   0.520   8.965

dielectric constant = 101.2028
dielectric constant without local fields = 143.7244
number of bands                                   150
   4   5.915 -11.654 -15.244   3.510   0.804  -0.244 -11.718  -0.065   5.850
   5   8.445  -9.702  -3.216  -5.853   0.810  -0.234  -9.189   0.514   8.959

dielectric constant = 101.2128
dielectric constant without local fields = 143.7244
```

```
number of bands                                    200
   4   5.915 -11.654 -15.244   3.509   0.803  -0.246 -11.719  -0.065   5.850
   5   8.445  -9.702  -3.216  -5.854   0.812  -0.231  -9.188   0.514   8.960
```

So that the computation using 100 bands can be considered converged within 0.01 eV.

5.6.8 Convergence on the dimension of the ϵ^{-1} matrix

Third, we check the convergence on the number of plane waves in the calculation of the screening. This will be done by defining six datasets, with increasing *ecuteps*:

```
ecuteps:?     3.0
ecuteps+?     1.0
```

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t68.in`, and modify the `t6x.files` file as usual. Edit the `t68.in` file, and take the time to examine it. Then, issue:

```
../../abinis < t6x.files >& t68.log &
```

This small job lasts about 25 secs on a PC PIV Intel 2.2 GHz.

Edit the output file. The number of bands used for the wavefunctions in the computation of the screening is mentioned in the fragments of output:

```
EPSILON^-1 parameters (EM1 file):
dimension of the eps^-1 matrix                     59
```

Gathering the macroscopic dielectric constant and GW energies for each number of bands, one gets:

```
dielectric constant = 102.1281
dielectric constant without local fields = 143.7244
dimension of the eps^-1 matrix                     59
   4   5.915 -11.654 -15.244   3.684   0.806  -0.241 -11.579   0.075   5.990
   5   8.445  -9.702  -3.216  -5.847   0.811  -0.232  -9.183   0.519   8.964

dielectric constant = 101.2712
dielectric constant without local fields = 143.7244
dimension of the eps^-1 matrix                    113
   4   5.915 -11.654 -15.244   3.559   0.804  -0.243 -11.680  -0.026   5.889
   5   8.445  -9.702  -3.216  -5.850   0.811  -0.233  -9.185   0.517   8.962

dielectric constant = 101.2649
dielectric constant without local fields = 143.7244
dimension of the eps^-1 matrix                    137
   4   5.915 -11.654 -15.244   3.535   0.804  -0.244 -11.699  -0.045   5.870
   5   8.445  -9.702  -3.216  -5.846   0.811  -0.232  -9.182   0.520   8.965

dielectric constant = 101.1764
dielectric constant without local fields = 143.7244
dimension of the eps^-1 matrix                    169
   4   5.915 -11.654 -15.244   3.516   0.804  -0.244 -11.714  -0.060   5.855
   5   8.445  -9.702  -3.216  -5.846   0.811  -0.233  -9.182   0.520   8.965

dielectric constant = 101.1384
dielectric constant without local fields = 143.7244
dimension of the eps^-1 matrix                    259
   4   5.915 -11.654 -15.244   3.517   0.804  -0.244 -11.713  -0.060   5.855
   5   8.445  -9.702  -3.216  -5.845   0.811  -0.232  -9.182   0.521   8.966
```

So that *npweps*=169 (*ecuteps*=6.0) can be considered converged within 0.01 eV.

At this stage, we know that for the screening computation, we need *ecutwfn*=4.0 *ecuteps*=6.0, *nband*=100.

Of course, until now, we have skipped the most difficult part of the convergence tests: the number of k–points. It is as important to check the convergence on this parameter, than on the other ones. However, this might be very time consuming, since the CPU time scales as the square of the number of k–points (roughly), and the number of k–points can increase very rapidly from one possible grid to the next denser one. This is why we will leave this out of the present tutorial, and consider that we already know a sufficient k–point grid, for the last calculation.

5.6.9 Calculation of the GW corrections for the band gap in Gamma

Now we try to perform a GW calculation for a real problem: the calculation of the GW corrections for the direct band gap of bulk Silicon in Gamma.

In directory `~ABINIT/Tutorial/Work6`, copy the file `../t69.in`, and modify the `t6x.files` file as usual. DO NOT EDIT IT NOW. Issue:

```
../../abinis < t6x.files >& t69.log &
```

This job lasts about 20 minutes on a PC PIV Intel 2.2 GHz. Because it is so long, it was worth to run it before the examination of the input file.

Now, you can examine it.

We need the usual part of the input file to perform a ground state calculation. This is done in dataset 1 and at the end we print out the density. We use a 4x4x4 FCC grid (so, 256 k–points in the full Brillouin Zone), shifted, because it is the most economical. It gives 10 k–points in the Irreducible part of the Brillouin Zone. However, this k–point grid does not contains the Gamma point, and, at present, one cannot perform calculations of the self–energy corrections for other k–points than those present in the grid of k–points in the KSS file.

Then in dataset 2 we perform a non self–consistent calculation to calculate the Kohn–Sham structure in a set of 19 k–points in the Irreducible Brillouin Zone. This set of k–points is also derived from a 4x4x4 FCC grid, but a NON–SHIFTED one. It has the same density of points as the 10 k–point set, but the symmetries are not used in a very efficient way. However, this set contains the Gamma point, which allows us to tackle the computation of the band gap at this point.

In dataset 3 we calculate the screening. The screening calculation is very time–consuming. So, we have decided to weaken a bit the parameters found in the previous convergence studies. Indeed, *ecutwfn* has been decreased from 4.0 to 3.6 . This is rather innocuous. Also, *nband* has been decreased from 100 to 25. This is a drastic change. The CPU time of this part is linear with respect to this parameter (or more exactly, with the number of conduction bands). Thus, the CPU time has been decreased by a factor of 4. Referring to our previous convergence study, we see that the absolute accuracy on the GW energies is now on the order of 0.2 eV only. However, the gap energy (difference between valence and conduction states) is likely correct within 0.02 eV.

Finally in dataset 4 we calculate the self–energy matrix element in Gamma, using the previously determined parameters.

You should obtain the following results:

```
k =    0.000   0.000   0.000
Band      E0  VxcLDA    SigX SigC(E0)      Z dSigC/dE  Sig(E)    E-E0       E
   4   5.915 -11.255 -12.425   0.861   0.771  -0.296 -11.493  -0.238   5.677
   5   8.445 -10.067  -5.858  -3.690   0.772  -0.296  -9.666   0.401   8.846

E^0_gap          2.530
E^GW_gap         3.169
DeltaE^GW_gap    0.639
```

So that the LDA energy gap in Gamma is about 2.53 eV, while the GW correction is about 0.64 eV, so that the GW band gap found is 3.17 eV.

One can compare now what have been obtained to what one can get from the literature.

EXP	3.40 eV	Landolt–Boernstein
LDA	2.57 eV	L. Hedin, Phys. Rev. 139, A796 (1965)
LDA	2.57 eV	M.S. Hybertsen and S. Louie, PRL 55, 1418 (1985)
LDA (FLAPW)	2.55 eV	N. Hamada, M. Hwang and A.J. Freeman, PRB 41, 3620 (1990)
LDA (PAW)	2.53 eV	B. Arnaud and M. Alouani, PRB 62, 4464 (2000)
LDA	2.53 eV	present work
GW	3.27 eV	M.S. Hybertsen and S. Louie, PRL 55, 1418 (1985)
GW	3.35 eV	M.S. Hybertsen and S. Louie, PRB 34, 5390 (1986)
GW	3.30 eV	R.W. Godby, M. Schlueter, L.J. Sham, PRB 37, 10159 (1988)
GW (FLAPW)	3.30 eV	N. Hamada, M. Hwang and A.J. Freeman, PRB 41, 3620 (1990)
GW (PAW)	3.15 eV	B. Arnaud and M. Alouani, PRB 62, 4464 (2000)
GW (FLAPW)	3.12 eV	W. Ku and A.G. Eguiluz, PRL 89, 126401 (2002)
GW	3.17 eV	present work

The values are spread over an interval of 0.2 eV. They depend on the details of the calculation. In the case of pseudopotential calculations, They depend of course on the pseudopotential used. However, a GW result is hardly meaningful beyond 0.1 eV, in the present state of the art.

Chapter 6

ABINIS Help: Help file for the main code of the ABINIT package

This document explains the i/o parameters and format needed for the main code (abinis) in the ABINIT package.

The new user is advised to read first the "new user's guide", before reading the present file. It will be easier to discover the present file with the help of the tutorial.

It is worthwhile to print this help file, for ease of reading.

When the user is sufficiently familiarized with ABINIT, the reading of the `~ABINIT/Infos/tuning` file might be useful. For response–function calculations using abinis, the complementary respfn help file `~ABINIT/Infos/respfn_help.html` is needed.

6.1 How to run the code

6.1.1 Introducing the files file

Given an input file (parameters described below) and the required pseudopotential files, the user must create a "files" file which lists names for the files the job will require, including the main input file, the main output file, root names for other input, output, or temporary files, and different pseudopotential file names.

The files file (called for example `ab.files`) could look like:

- `ab_in`;
- `ab_out`;
- `abi`;
- `abo`;
- `tmp`;
- `14si.psp`.

In this example:

- the main input file is called "ab_in",

- the main output will be put into the file called "ab_out",
- the name of input wavefunctions (if any) will be built from the root abi (namely `abi_WFK`, see later),
- the output wavefunctions will be written to `abo_WFK`. Other output files might be build from this root,
- the temporary files will have a name that use the root "tmp" (for example `tmp_STATUS`),
- the pseudopotential needed for this job is "14si.psp".

Other examples are given in the `~ABINIT/Test_fast` directory. The maximal length of names for the main input or output files is presently 132 characters. It is 112 characters for the root strings, since they will be supplemented by different character strings.

If you follow the tutorial, you should go back to the tutorial window now.

6.1.2 Running the code

The main executable files are called abinis (sequential version), or abinip (parallel version). In the present help file, we will concentrate on the sequential version. There is a brief introduction to the use of the parallel version in the `~ABINIT/Infos/paral_use` file. Supposing that the "files" file is called `ab.files`, and that the executable is placed in your working directory, abinis is run interactively (in Unix) with the command

```
abinis < ab.files >& log
```

or, in the background, with the command

```
abinis < ab.files >& log &
```

where standard out and standard error are piped to the log file called "log" (piping the standard error, thanks to the "&" sign placed after "¿" is really important for the analysis of eventual failures, when not due to ABINIT, but to other sources, like disk full problem ...). The user can specify any names he/she wishes for any of these files. Variations of the above commands could be needed, depending on the flavor of UNIX that is used on the platform that is considered for running the code.

If you follow the tutorial, you should go back to the tutorial window now.

6.1.3 The underlying theoretical framework and algorithms

See the "bibliography" file.

The methods employed in this computer code to solve the electronic structure problem are described in part in different review papers as well as research papers. The code is an implementation of the Local Density Approximation to the Density Functional Theory, based upon a plane wave basis set and separable pseudopotentials. The iterative minimization algorithm is a combination of fixed potential preconditioned conjugate gradient optimization of wavefunction and a choice of different algorithms for the update of the potential, one of which is a potential–based conjugate gradient algorithm.

The representation of potential, density and wavefunctions in real space will be done on a regular 3D grid of points. Its spacing will be determined by the cut–off energy (see the input variable "ecut") of the planewave basis in reciprocal space. This grid of points will also be the starting point of Fast Fourier Transforms between real and reciprocal space. The number of such points, called "ngfft", should be sufficiently large for adequate representation of the functions, but not too large, for reason of computational efficiency. The trade–off between accuracy and computational efficiency is present in many places in the code, and addressed briefly at the end of the present help file.

We recommend a good introduction to many different concepts valid for this code, available in a Reviews of Modern Physics article, "Iterative minimization techniques for *ab initio* total–energy calculations: molecular dynamics and conjugate gradients", M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos, Rev. Mod. Phys. 64, 1045–1097 (1992). This paper does NOT reflect the present status of the code. ABINIT is closer in spirit to the paper of of Kresse and Furthmuller, see the bibliography list (except that it does not use ultrasoft pseudopotentials, and that response functions have been implemented in ABINIT.)

6.2 The input file

6.2.1 Format of the input file

Note that this input file was called `ab_in` in the example of section 1.1. We first explain the content of the input file without use of the "multi–dataset" possibility (that will be explained in section 3.3).

The parameters are input to the code from a single input file. Each parameter value is provided by giving the name of the input variable and then placing the numerical value(s) beside the name, separated by one or more spaces. Depending on the input variable, the numerical value may be an integer or a real number (internal representation as double precision number), and may actually represent an array of values. If it represents an array, the next set of numbers separated by spaces are taken as the values for the array.

Do NOT separate a minus sign from the number to which it applies.

Do NOT use tabs.

NOTE THAT NO LINE OF THE INPUT FILE MAY EXCEED 132 CHARACTERS. That is, only the first 132 characters of each line of the input file will be read and parsed for input variables and their values.

The names of all the parameters can be found in the input variables file. The definitions of all the parameters can be found in:

- Basic variables, VARBAS;
- Developpement variables, VARDEV;
- Geometry builder + symmetry related variables, VARGEO;
- Ground–state calculation variables, VARGS;
- Files handling variables, VARFIL;
- Parallelisation variables, VARPAR;
- Response Function variables, VARRF;
- Structure optimization variables, VARRLX.

In the actual input file, these parameters may be given in any order desired and more than one may be given per line. Spaces are used to separate values and additional spaces are ignored. An as example of input, the parameter for length scales is called "acell" and is an array acell(3) for the lengths of the primitive translations in bohr atomic units. To input a typical Si diamond lattice one would have the line

```
acell 10.25311 10.25311 10.25311
```

in the input file. This may equivalently be written

```
acell 3*10.25311
```

and will still be parsed correctly.

Multiple spaces are ignored, as is any text which does not contain the character strings which correspond to some input parameters. In case of arrays, only the needed numbers will be considered, and the eventual numbers after those needed will also be ignored. For example,

```
natom 3 # This gives the number of atoms
typat 1 1 2 2 3 # typat(1:natom) gives the type of each atom : only
                # the three first data are read, since natom=3
```

A given variable is identified by the parser by having at least one blank before it and after it (again, multiple blanks are irrelevant). ABINIT has also some (very limited) interpretor capabilities:

- It can identify one slash sign (/) being placed between two numbers (without a separating blank) as being the definition of a fraction (e.g. 1/3 will be interpreted as 0.33333333333333d0);
- It can identify sqrt(...) or –sqrt(...) as being the definition of a square root, when applied to one valid number — also without a separating blank — (e.g. –sqrt(0.75) will be interpreted as –0.8660254038d0);
- Note, however, that these capabilities are NOT recursive, and cannot be used one with the other (e.g. sqrt(3/4) is invalid).

To include comments it is recommended that they be placed to the right of the comment characters `#` or `!`; anything to the right of a "#" or a "!" on any line is simply ignored by the parser. Additional text, not preceeded by a "#" or a "!" would not otherwise cause trouble unless the text inadvertantly contained character strings which were the same as variable names (e.g. "acell"). The characters "#" or "!" can also be used to "store" old values of variables or place anything else of convenience into the file in such a way as to be ignored by the parser when the data is read. Case is irrelevant as the entire input string is mapped to upper case before parsing, to remove case sensitivity. More than one parameter per line may be given. If a given parameter name is given more than once in the input file, an error message is printed, and the code stops.

If you follow the tutorial, you should go back to the tutorial window now.

6.2.2 More about ABINIT input variables

In each section of the ABINIT input variables files, a generic information on the input variable is given: a mnemonics, some "characteristics", the variable type, and the default. Then, follows the description of the variable.

The mnemonics is indicated when available.

The "characteristics" can be of different types: DEVELOP, RESPFN, GEOMETRY BUILDER, SYMMETRISER, SYMMETRY FINDER, NO MULTI, EVOLVING, ENERGY, LENGTH. We now explain each of these classes.

'DEVELOP' refers to input variables that are not used in production runs, but only during development time. For non developers, it is strongly advised to skip them.

Some input variables are related to response function features, and are indicated 'RESPFN'. Detailed explanations related to response function features are to be found in the complementary respfn help file `~ABINIT/Infos/respfn_help.html`. The initials RF are used for 'response function', and non–response–function are often referred to as GS (for ground–state), although this latter designation is not really satisfactory.

There are also parameters related to the geometry builder, a preprocessor of the input file, aimed at easing the work of the user when there are molecules to be manipulated (rotation and translation), or group of atoms to be repeated. The indication 'GEOMETRY BUILDER' is given for them. These can also be skipped for the first few steps in the use of the code. Indeed, it should be easy to set up the geometry of systems with less than 20–40 atoms without this

geometry builder. Even for larger systems, its functionalities could eventually be of no help. For a step–to–step description of this geometry builder, look at the variable 'nobj'.

Alternatively to the geometry builder, there is also a symmetriser. It allows to generate the full set of atoms in the primitive cell from the knowledge of the symmetry operations and the atoms in the asymetric cell. It also allows to generate the symmetry operations from the knowledge of the number of the space group according to the international crystallographic tables. The indication 'SYMMETRISER' is given for the variables related to its use. Look at the variable 'spgroup'. You may find in the space group help file the crystallographic equivalence of the parameters belonging to the symmetriser.

Still as an alternative to the geometry builder and the symmetriser, if all the coordinates of the atoms are given, the code is able to deduce all symmetry operations leaving the lattice and atomic sublattices invariant, see 'SYMMETRY FINDER'.

Most of the variables can be used in the multi–dataset mode (see section 3.3), but those that must have a unique value throughout all the datasets are signaled with the indication 'NO MULTI'.

Most of the input variables do not change while a run is performed. Some of them, by contrast, may evolve, like the atomic positions, the atomic velocities, the cell shape, and the occupation numbers. Their echo, after the run has proceeded, will of course differ from their input value. They are signaled by the indication 'EVOLVING'.

The use of the atomic unit system (e.g. the Hartree for energy, about 27.211 eV, and the Bohr for lengths about 0.529 Angstroms) is strictly enforced within the code. However, the dimension of some input variables can be specified and read correctly. At present, this applies to two types of variables: those that have the dimension of an energy, and those that have a dimension of length. The first class of variables have the characteristics 'ENERGY', and can be specified in atomic units (Hartree), or electron–volts, or Rydbergs, or even Kelvin. The second class of variables have the characteristics 'LENGTH', and can be specified in atomic units (Bohr) and angstrom. The abinit parser recognize a dimension if it is specified after the list of numbers following the input variable keyword, in the input file. The specification can be upper or lower case, or a mix thereof. Here is the list of recognized chains of characters:

- 'Ry ' ⇒ Rydberg (for energies);
- 'eV '⇒ electron–volts (for energies);
- 'K ' ⇒ Kelvin (for energies);
- 'Angstr ... ' ⇒ Angstrom (for lengths).

Except in the case of 'Angstr', the abbreviation must be used (i.e. 'Rydberg' will not be recognized presently). Other character chains, like "au" (for atomic units) or "Hartree", or "Bohr" are not recognized, but make the parser choose (by default) atomic units, which is the correct behaviour. Example:

```
acell 8 8 8 angstrom
ecut 8 Ry
tsmear 1000 K
```

or

```
acell 3*10 Bohr  ecut 270 eV  tsmear 0.01
```

The use of the atomic units is mandatory for other dimensioned input variables, like the tolerance on forces (*toldff*), parameters that define an 'object' (*objaax*, *objbax*, *objatr*, *objbtr*), and the initial velocity of atoms (vel — if needed).

The initial atomic positions can be input in Bohr or Angstrom through 'xcart', but also, independently, in Angstrom through 'xangst', or even in reduced coordinates, through 'xred'. Reduced cartesian coordinates must be used for the eventual translations accompanying symmetry operations (*tnons*).

In addition to giving the input variables, the input file can be useful for another purpose: placing the word "exit" on the top line will cause the job to end smoothly on the very next iteration. This functions because the program closes and reopens the input file on every iteration and checks the top line for the keyword "exit". THE WORD MUST BE PLACED WITH SPACES (BLANKS) ON BOTH SIDES. Thus placing exit on the top line of the input file WHILE THE JOB IS ALREADY RUNNING will force the job to end smoothly on the very next iteration. On some machines, this does not work always (we do not know why ...). Another possibility is offered : one can create a file named "abinit.exit" in the directory where the job was started. The code should also smoothly end. In both cases, the stop is not immediate. It can take a significant fraction (about 20% at most) of one SCF step to execute properly the instruction still needed.

If you follow the tutorial, you should go back to the tutorial window now.

6.2.3 The multi–dataset mode

Until now, we have assumed that the user wants to make computations corresponding to one set of data: for example, determination of the total energy for some geometry, with some set of plane waves and some set of k–points.

It is often needed to redo the calculations for different values of some parameter, letting all the other things equal. As typical examples, we have convergence studies needed to determine which cut–off energy gives the needed accuracy. In other cases, one makes chains of calculations in order to compute the band structure: first a self–consistent calculation of the density and potential, then the eigenenergy computation along different lines.

For that purpose, the multi–dataset mode has been implemented.

It allows the code to treat, in one run, different sets of data, and to chain them. The number of datasets to be treated is specified by the variable ndtset, while the indices of the datasets (by default 1, 2, 3, and so on) can be eventually provided by the array *jdtset*.

For each dataset to be treated, characterized by some index, each input variable will determined by the following rules (actually, it is easier to understand when one looks at examples, see below):

1. ABINIT looks whether the variable name (e.g. *ecut*), appended with the index of the dataset (e.g. *jdtset*=2), exists (e.g. "ecut2") . It will take the data that follows this keyword, if it exist;

2. If this modified variable name does not exist, it will look whether a metacharacter, a series or a double–loop data set has been defined, see sections 3.4 or 3.5;

3. If the variable name appended with the index of the dataset does not exist, and if there is no series nor double–loop dataset for this keyword, it looks for an occurence of the variable name without any index appended, and take the corresponding data. (This corresponds to the single dataset mode);

4. If such occurences do not exist, it takes the default value. (Also, similar to the single dataset mode).

1st example.

```
ndtset   2
 acell   8 8 8
  ecut1  10
  ecut2  15
```

means that there are 2 datasets: a first in which

```
acell 8 8 8  ecut 10
```

has to be used, and a second in which

```
acell 8 8 8  ecut 15
```

has to be used.

2nd example.

```
ndtset 2     jdtset 4 5

acell   8 8 8
acell5 10 10 10
ecut1  10
ecut2  15
ecut3  20
ecut4  25
ecut5  30
```

this means that there are still two datasets, but now characterized by the indices 4 and 5, so that the first run will use the generic "acell", and "ecut4":

```
acell 8 8 8 ecut 25
```

and the second run will use "acell5" and "ecut5":

```
acell 10 10 10 ecut 30
```

Note that *ecut1*, *ecut2* and *ecut3* are not used.

6.2.4 Defining a series

Rules (2) is split in three parts: (2a), (2b) and (2c). Series relate with (2b):

> (2b) If the variable name appended with the index of the dataset does not exist, the code looks whether a series has been defined for this keyword.

There are two kinds of series:

- arithmetic series (constant increment between terms of the series);
- geometric series (constant ratio between terms of the series).

The first term of the series is defined by the keyword appended with a colon (e.g. *ecut:*), while the increment of an arithmetic series is defined by the keyword appended with a plus (e.g. *ecut+*), and the factor of a geometric series is defined by the keyword appended with a times (e.g. *ecut**).

If the index of the dataset is 1, the first term of the series is used, while for index N, the appropriate input data is obtained by considering the Nth term of the series.

3rd example

```
ndtset 6
ecut1 10
ecut2 15
ecut3 20
ecut4 25
ecut5 30
ecut6 35
```

is equivalent to

```
ndtset 6 ecut: 10 ecut+ 5
```

In both cases, there are six datasets, with increasing values of *ecut*.

6.2.5 Defining a double loop dataset

To define a double loop dataset, one has first to define the upper limit of two loop counters, thanks to the variable *udtset*. The inner loop will execute from 1 to *udtset(2)*, and the outer loop will execute from 1 to *udtset(1)*. Note that the largest value for *udtset(1)* and *udtset(2)* is 9 presently.

The value of *ndtset* must be coherent with *udtset* (it must equal the product *udtset(1)*udtset(2)*).

A dataset index is created by the concatenation of the outer loop index and the inner loop index. For example, if *udtset(1)* is 2 and *udtset(2)* is 4, the index will assume the following values: 11, 12, 13, 14, 21, 22, 23, and 24.

Independently of the use of *udtset*, rules (2a) and (2c) will be used to define the value of an input variable:

> (2a) The question mark "?" can be used as a metacharacter, replacing any digit from 1 to 9, to define an index of a dataset. For example, *ecut?1* means that the input value that follows it can be used for ecut for the datasets 01, 11, 21, 31, 41, 51, 61, 71, 81, and 91.
>
> (2c) If the variable name appended with the index of the dataset does not exist, the code looks whether a double–loop series has been defined for this keyword. Series can be defined for the inner loop index or the outer loop index. Two signs will be appended to the variable name (instead of one in the simple series case). One of these signs must be a question mark "?", again used as a metacharacter able to assume the values 1 to 9. If it is found in the first of the two positions, it means that the series does not care about the outer loop index (so the values generated are equal for all outer loop index values). If it is found in the second of the two positions, the series does not care about the inner loop index. The other sign can be a colon, a plus or a times, as in the case of the series defined in (2a), with the same meaning.

Rule (1) has precedence over them, they have precedence over rules (3) or (4), rule (2a) has precedence over rules (2b) or (2c) and the two latter cannot be used simultaneously for the same variable.

4th example

```
ndtset 6    udtset 2 3
acell1?  10 10 10
acell2?  15 15 15
ecut?: 5    ecut?+ 1
```

is equivalent to

```
ndtset 6     jdtset 11 12 13  21 22 23
acell11  10 10 10     ecut11 5
acell12  10 10 10     ecut12 6
acell13  10 10 10     ecut13 7
acell21  15 15 15     ecut21 5
acell22  15 15 15     ecut22 6
acell23  15 15 15     ecut23 7
```

More examples can be found in the directory `Test_v1`, cases 59 and later.

6.2.6 File names in the multi–dataset mode

The root names for input and output files (potential, density, wavefunctions and so on) will receive an appendix : '_DS' followed by the index of the dataset. See section 4.

The 'get' variables can be used to chain the calculations.

Until now, there are eight of them: *getwfk*, *getwfq*, *getddk*, *get1wf*, *getden*, *getcell*, *getxred* and *getxcart*.

- *getwfk* allows to take the output wavefunctions of a previous dataset and use them as input wavefunctions;
- *getwfq*, *getddk* and *get1wf* do similar things for response function calculations;
- *getden* does the same for the density; *getcell* does the same for *acell* and *rprim*;
- *getxred* and *getxcart* do the same for the atomic positions, either in reduced coordinates, or in cartesian coordinates.

The different variables corresponding to each dataset are echoed using the same indexing convention as for the input step. For the last echo of the code variables, some output variables are also summarized, using the same conventions:

- **etotal** (total energy);
- **fcart** (cartesian forces);
- **strten** (the stress tensor).

If you follow the tutorial, you should go back to the tutorial window now.

6.3 The "files" file

Note: *This "files" file is called* `ab.files` *in section 1.1.*

Contains the file names or root names needed to build file names. These are listed below: there are 5 names or root names for input, output and temporaries, and then a list of pseudopotentials. These names may be provided from unit 05 interactively during the run but are more typically provided by piping from a file in unix (the "files" file).

ab_in: Filename of file containing the input data, described in the preceeding sections.

ab_out: Filename of the main file in which formatted output will be placed (the main output file). Error messages and other diagnostics will NOT be placed in this file, but sent to unit 06 (terminal or log file); the unit 06 output can be ignored unless something goes wrong. The code repeats a lot of information to both unit 06 and to the main output file. The unit 06 output is intended to be discarded if the run completes successfully, with the main output file keeping the record of the run in a nicer looking format.

abi: The other files READ by the code will have a name that is constructed from the root "abi". This apply to optionally read wavefunction, density or potential files. In the multi–dataset mode, this root will be complemented by '_DS' and the dataset index. The list of possible input files, with their name created from the root 'abi' is the following (a similar list exist when '_DS' and the dataset index are appended to 'abi'):

- **abi_WFK**
  filename of file containing input wavefunction coefficients created from an earlier run (with *nqpt*=0). Will be opened and read if *irdwfk* is 1. The wavefunction file is unformatted and can be very large. **Warning**: in the multi–dataset mode, if *getwfk* is non–zero, a wavefunction file build from **abo** will be read.
- **abi_WFQ**
  filename of file containing input wavefunction coefficients created from an earlier run (with *nqpt*=1), as needed for response function salculations. The wavefunction file is unformatted and can be very large. **Warning**: in the multi–dataset mode, if *getwfk* is non–zero, a wavefunction file build from **abo** will be read.
- **abi_1WFxx**
  filename of file containing input first–order wavefunctions created from an earlier RF run. xx is the index of the perturbation.

- **abi_DEN**
  filename of file containing density created from an earlier run. See explanations related to negative values of *iscf*. This file is also unformatted. **Warning**: in the multi dataset mode, if *getwfk* is non–zero, a density file build from **abo** will be read.

- **abi_HES**
  filename of file containing an approximate hessian, for eventual (re)initialisation of Broyden minimisation. See `brdmin.f` routine. The use of *restartxf* is preferred.

abo: Except "ab_out" and "log", the other files WRITTEN by the code will have a name that is constructed from the root "abo". This apply to optionally written wavefunction, density, potential, or density of states files. In the multi–dataset mode, this root will be complemented by '_DS' and the dataset index. Also in the multi–dataset mode, the root "abo" can be used to build the name of input files, thanks to the 'get' variables. The list of possible input files, with their name created from the root 'abo' is the following (a similar list exists when '_DS' and the dataset index are appended to 'abo'):

- **abo_WFK**
  Filename of file containing output wavefunction coefficients, if *nqpt*=0. The wavefunction file is unformatted and can be very large.

- **abo_WFQ**
  Same as **abo_WFK**, but for the case *nqpt*=1. The wavefunctions are always output, either with the name **abo_WFK**, or with the name **abo_WFQ**.

- **abo_1WFxx**
  Same as **abo_WFK**, but for first–order wavefunctions, xx is the index of the perturbation, see the section 6.3 of the `respfn_help.html` file.

- **abo_DDB**
  The derivative database, produced by a response–function dataset, see the section 6.5 of the `respfn_help.html` file.

- **abo_DEN**
  Filename of file containing density, in the case *ionmov*=0. See the keyword *prtden*. This file is unformatted, but can be read by `cut3d`.

- **abo_TIMx_DEN**
  Filenames of files containing density, in the case *ionmov*≠0. The value of "x" after "TIM" is described hereafter. See the keyword *prtden*. This file is unformatted, but can be read by `cut3d`.

- **abo_POT**
  Filename of file containing Kohn–Sham potential See the keyword *prtopt*. This file is unformatted, but can be read by `cut3d`.

- **abo_TIMx_POT**
  Filenames of files containing Kohn–Sham potential in the case *ionmov*≠0. The value of "x" after "TIM" is described hereafter. See the keyword *prtopt*. This file is unformatted, but can be read by `cut3d`.

- **abo_DOS**
  Filename of file containing density of states. See the keyword prtdos. This file is formatted.

- **abo_TIMx_DOS**
  Filenames of files containing the density of states in the case *prtdos*=2 and *ionmov*=1 or 2. The value of "x" after "TIM" is described hereafter. See also the keyword *prtdos*. This file is formatted.

- **abo_GEO**
  Filename of file containing the geometrical analysis (bond lengths and bond angles) in the case *ionmov*=0. See the keyword *prtgeo*. This file is formatted.

- **abo_TIMx_GEO**
  Filenames of files containing the geometrical analysis (bond lengths and bond angles) in the case *ionmov*=1 or 2. The value of "x" after "TIM" is described hereafter. See also the keyword *prtgeo*. This file is formatted.

- **abo_CML.xml**
  filename of file containing the Chemical Markup Language description of the system (crystallographic data, symmetry data, atomic symbols and reduced coordinates) in the case *ionmov*=0. See the keyword *prtcml*. This file is formatted.

- **abo_TIMx_GEO**
  Filenames of files containing the Chemical Markup Language description of the system (crystallographic data, symmetry data, atomic symbols and reduced coordinates) in the case *ionmov*=1 or 2. The value of "x" after "TIM" is described hereafter. See also the keyword *prtcml*. This file is formatted.

- **abo_STO**
  Filename of file containing output wavefunction coefficients, if *nbandkss* $\neq$ 0. This wavefunction file is unformatted and can be very large. Its purpose is to start a GW calculation using M. Torrent's code. A different format than for **abo_WFK** is used, see the file `~ABINIT/Infos/format_STO`.

When *ionmov* $\neq$ 0, the **POT**, **DEN**, **GEO**, or **CML.xml** files are output each time that a SCF cycle is finished. The "x" of **TIMx** aims at giving each of these files a different name. It is attributed as follows:

- case *ionmov*==1: there is an initialization phase, that takes 4 calls to the SCF calculation. The value of x will be A, B, C, and D. Then, x will be 1, 2, 3 ... , actually in agreement with the value of *itime* (see the keyword *ntime*);

- other *ionmov* cases: the initialisation phase take only one SCF call. The value of x will be 0 for that call. Then, the value of x is 1, 2, 3 ... in agreement with the value of *itime* (see the keyword *ntime*).

tmp: The temporary files created by the codes will have a name that is constructed from the root "tmp". tmp should usually be chosen such as to give access to a disk of the machine that is running the job, not a remote (NFS) disk. Under Unix, the name might be something like `/tmp/user_name/temp`. The most important temporary files, with their name created from the root "tmp" is the following:

- **tmp_FFT**
  not created if *mffmem*==1, contains a few arrays defined in real space on the FFT grid.

- **tmp_KG**
  not created if *mkmem*=*nkpt*, contains the data on G vectors inside the sphere around the different k–points.

- **tmp_KGS**
  created if *iprcel*$\neq$0, contains the data on G vectors inside the sphere around the different k–points, for the computation of the susceptibility.

- **tmp_WF1** and

- **tmp_WF2**
  not created if *mkmem*=*nkpt*, contains the wavefunctions in the process of the calculation.

- **tmp_STATUS**
  gives the status of advancement of the calculation, and is updated very frequently.

psp1: filename of first pseudopotential input file. The pseudopotential data files are formatted. There must be as many filenames provided sequentially here as there are types of atoms in the system, and the order in which the names are given establishes the identity of the atoms in the unit cell. (psp2, psp3, ...)

If you follow the tutorial, you should go back to the tutorial window now.

6.4 The pseudopotential files

The following section describes the file structure used for the pseudopotential files with different formats. Actually, no real understanding of these files is needed to run the code, but for different other reasons, it might be useful to be able to understand the file structures. Different format are possible (labelled 1 to 6 presently) The associated internal variable is called *pspcod*. Example of use are found in `~ABINIT/Test_v1`. Informations on the file structure can be found in the `~ABINIT/Infos/Psp_infos` directory.

- *pspcod*=1: Troullier–Martins pseudopotentials, generated by D. C. Allan and A. Khein, see `~ABINIT/Infos/Psp_infos/psp1.info`;
- *pspcod*=2: Goedecker–Teter–Hutter (GTH) pseudopotentials. See Phys. Rev. B 54, 1703 (1996) if needed;
- *pspcod*=3: Hartwigsen–Goedecker–Hutter pseudopotentials. See Phys. Rev. B 58, 3641 (1998) if needed, and the file `~ABINIT/Infos/Psp_infos/psp3.info`.
- *pspcod*=4 or 5: old format pseudopotentials, see `~ABINIT/Infos/Psp_infos/psp45.info`.
- *pspcod*=6: pseudopotentials from the fhi98pp code, see `~ABINIT/Infos/Psp_infos/psp6.info`.

6.5 The different output files

Explanation of the output from the code.

Output from the code goes to several places listed below.

6.5.1 The log file

The "log" file (this is the standard UNIX output file, and corresponds to Fortran unit number 06): a file which echoes the values of the input parameters and describes various steps of the calculation, typically in much more detail than is desired as a permanent record of the run. This log file is intended to be informative in case of an error or for a fuller description of the run. For a successful run the user will generally delete the log file afterwards. There are four types of exception messages: ERROR, BUG, WARNING and COMMENT messages.

ERROR and **BUG** messages cause the code to stop, immediately, or after a very small delay. An **ERROR** is attributed to the user, while a **BUG** is attributed to the developer.

A **WARNING** message indicates that something happened that is not as expected, but this something is not so important as to make the code stop. A **COMMENT** message gives some information to the user, concerning something unusual. None of them should appear when the run is completely normal.

After a run is completed, always have a look at the end of the log file, to see whether an **ERROR** or a **BUG** occurred.

Also, the code gives the number of **WARNING** or **COMMENT** it issued. It is advised to read at least the **WARNING** messages, during the first month of ABINIT use.

If you follow the tutorial, you should go back to the tutorial window now.

6.5.2 The main output file

The **main output file** is a formatted output file to be kept as the permanent record of the run.

Note that it is expected not to exist at the beginning of the run: If a file with the name specified in the "files" file already exists, the code will generate, from the given one, another name, appended with **.A**. If this new name already exists, it will try to append **.B**, and so on, until **.Z**. Then, the code stops, and asks you to clean the directory.

The **main output file** starts with a heading:

- version number and specified platform;
- copyright notice and distribution licence;
- date;
- echo of "files" file (except pseudopotential name).

Then, for each dataset, it reports the point symmetry group and Bravais lattice, and the expected memory needs. It echoes the input data, and report on checks of data consistency for each dataset.

If you follow the tutorial, you should go back to the tutorial window now.

6.5.3 More on the main output file

Then, for each dataset, the real computation is done, and the code will report on some initialisations, the SCF convergence, and the final analysis of results for this dataset. Each of these phases is now described in more details.

The code reports:

- the real and reciprocal space translation vectors (Note: the definition of the reciprocal vector is such that $R_i G_j = delta_{ij}$);
- the volume of the unit cell;
- the ratio between linear dimension of the FFT box and the sphere of plane waves, called "boxcut";
- It must be above 2 for exact treatment of convolutions by FFT. *ngfft* has been automatically chosen to give a boxcut value larger than 2, but not much larger, since more CPU time is needed for larger FFT grids;
- the code also mention that for the same FFT grid you might treat (slightly) larger *ecut* (so, with a rather small increase of CPU time);
- the heading for each pseudopotential which has been input;
- from the `inwffil` subroutine, a description of the wavefunction initialization (random number initialization or input from a disk file), that is, a report of the number of planewaves (*npw*) in the basis at each k–point;
- from the `setup2` subroutine, the average number of planewaves over all k–points is reported in two forms, arithmetic average and geometric average.

Until here, the output of a ground–state computation is identical to the one of a response–function calculation. See the `respfn_help` document for the latter, especially section 6.2.

Next the code reports information for each SCF iteration:

- the iteration number;
- the total energy (Etot) in Hartree;

- the change in Etot since last iteration (deltaE);
- the maximum squared residual residm over all bands and k–points (residm - the residual measures the quality of the wavefunction convergence);
- the squared residual of the potential in the SCF procedure (vres2);
- the maximum change in the gradients of Etot with respect to fractional coordinates (diffor, in Hartree);
- the rms value of the gradients of Etot with respect to fractional coordinates (maxfor, in hartree). *The latter two are directly related to forces on each atom.*
- Then comes an assessment of the SCF convergence: the criterion for fulfillment of the SCF criterion (defined by *toldfe*, *toldff*, *tolwfr* or *tolvrs*) might be satisfied or not ...
- Then the stresses are reported.

This ends the content of a fixed atomic position calculation.
Many such blocks can follow.
When the atomic positions have been eventually relaxed, according to the value of *ntime*, the code output more information:

- The squared residuals for each band are reported, k–point by k–point.
- Then the fractional or reduced coordinates are given,
- followed by the energy gradients,
- followed by the cartesian coordinates in Angstroms,
- followed by the cartesian forces in Hartree/Bohr and eV/Angstrom.
- Also are given the rms force (**frms**) and the maximum absolute value of any force component (**max**).
- Next are the length scales of the unit cell in Bohr and in Angstroms.
- Next are the eigenvalues of each band for each k–point, in eV or hartree or both depending on the choice of *enunit*.
  NOTE that the average electrostatic potential of a periodically repeated cell is UNDEFINED.
  In the present implementation, the average Hartree potential and local potential are imposed to be zero, but not the average exchange–correlation potential. This definition gives some meaning to the absolute values of eigenenergies, thanks to Janak's theorem: they are derivatives of the total energy with respect to occupation number. Indeed, the $G = 0$ contributions of the Hartree, local potential and ion–ion to the total energy is independent of the occupation number in the present implementation. With this noticeable exception, one should always work with **differences** in eigenenergies, as well as **differences** between eigenenergies and the potential. For example, the absolute eigenenergies of a bulk cell should not be used to try to predict a work function. The latter quantity should be obtained in a supercell geometry, by comparing the Fermi energy in a slab and the potential in the vacuum in the same supercell.
- Next are the minimum and maximum values for charge density, and next smaller or larger values (in order to see degeneracies).
- Next are the components of the total energy broken down into:
  - kinetic,

  - Hartree,
  - exchange and correlation (xc),
  - local pseudopotential,
  - nonlocal pseudopotential,
  - local pseudopotential "core correction", and
  - Ewald energies.

- Next is the stress tensor, $(1/uc_{vol})d(E_{tot})/d(strain_{a,b})$, for E_{tot} = total energy per unit cell and a, b are x, y or z components of strain. The stress tensor is given in Cartesian coordinates in Hartree/Bohr^3 and GPa. The basics of the stress tensor are described in O. H. Nielsen and Richard M. Martin, see the bibliography file.

Having finished all the calculations for the different datasets, the code echoes the parameters listed in the input file, using the latest values e.g. for *xred*, *vel*, and *xcart*, and supplement them with the values obtained for the total energy, the forces and stresses, as well as occupation numbers. The latter echoes are very convenient for a quick look at the result of calculation!

This is followed finally by the timing output: both "cpu" time and "wall clock" time as provided by calls within the code. The total cpu and wall clock times are reported first, in seconds, minutes, and hours for convenient checking at a glance. Next are the cpu and wall times for the principal time–consuming subroutine calls, each of which is independent of the others. The sum of these times usually accounts for about 90% of the run time.

The main subroutines, for BIG jobs, are

1. fourwf: the subroutine which performs the fast fourier transform for the wavefunctions;
2. fourdp: the subroutine which performs the fast fourier transform related to density and potential;
3. rhohxc: computes the Hartree and exchange–correlation energy and potential and sometimes derivative of potential; only the XC timing is reported, excluding time connected to the FFTs: `xc:pot/=fourdp`.
4. nonlop: computes $< G|V_{\text{non–local}}|C >$ the matrix elements of the nonlocal pseudopotential;
5. projbd: Gram–Schmidt orthogonalization.

In case of small jobs, other (initialisation) routines may take a larger share, and the sum of the times for the principal time–consuming subroutine calls will not make 90% of the run time ...

If the long printing option has been selected (*prtvol*=1), the code gives much more information in the whole output file. These should be rather self–explanatory, usually. Some need more explanation.

In particular the cpu and wall times for major subroutines which are NOT independent of each other; for example vtorho conducts the loop over k–points and calls practically everything else. In case of a ground state calculation, at fixed atomic positions, these subroutines are

1. **abinit**: the main routine;
2. **driver**: select ground state or response calculations;
3. **gstate**: the driver of the ground state calculations;
4. **scfcv**: the SCF cycle driver;
5. **vtorho**: compute the density from the potential (it includes a loop over spins and k–points);
6. **vtowfk**: compute the wavefunctions at one particular k–point (includes a non self consistent loop, and a loop over bands);

7. **cgwf**: optimize one wavefunction in a fixed potential;
8. **getghc**: computes $< G|H|C >$, that is, applies the Hamiltonian operator to an input vector.

If you follow the tutorial, you should go back to the tutorial window now.

6.5.4 The header

The **wavefunction files**, **density files**, and **potential files** all begins with the same records, called the "header". This header is treated using a hdr_type datastructure inside ABINIT. There are dedicated routines inside ABINIT for initializing a header, updating it, reading the header of an unformatted disk file, writing a header to an unformatted disk file, echoing a header to a formatted disk file, cleaning a header datastructure.

The header is made of 4+*npsp* unformatted records, obtained by the following Fortran90 instructions (format 4.1):

```
 write(unit=header) codvsn,headform,fform
 write(unit=header) bantot,date,intxc,ixc,natom,ngfft(1:3),&
& nkpt,nspden,nspinor,nsppol,nsym,npsp,ntypat,occopt,pertcase,&
& ecut,ecutsm,ecut_eff,qptn(1:3),rprimd(1:3,1:3),stmbias,tphysel,tsmear
 write(unit=header) istwfk(1:nkpt),nband(1:nkpt*nsppol),&
& npwarr(1:nkpt),so_typat(1:ntypat),symafm(1:nsym),symrel(1:3,1:3,1:nsym),&
& typat(1:natom),&
& kpt(1:3,1:nkpt),occ(1:bantot),tnons(1:3,1:nsym),znucltypat(1:ntypat)
 do ipsp=1,npsp
! (npsp lines, 1 for each pseudopotential ; npsp=ntypat, except if
! alchemical pseudo-atoms)
  write(unit=unit) title,znuclpsp,zionpsp,pspso,pspdat,pspcod,pspxc
 enddo
!(final record: residm, coordinates, total energy, Fermi energy)
 write(unit=unit) residm,xred(1:3,1:natom),etotal,fermie
```

where the type of the different variables is:

```
character*6 :: codvsn
integer :: headform,fform
integer :: bantot,date,intxc,ixc,natom,ngfft(3),nkpt,
nspden,nspinor,nsppol,nsym,ntypat,occopt,pertcase
double precision :: acell(3),ecut,ecutsm,ecut_eff,qptn(3),rprimd(3,3),&
& tphysel,tsmear
integer :: istwfk(nkpt),nband(nkpt*nsppol),npwarr(nkpt),so_typat(ntypat),&
& stmbias,&
& symafm(nsym),symrel(3,3,nsym),typat(natom)
double precision :: kpt(3,nkpt),occ(bantot),tnons(3,nsym),znucltypat(ntypat)
character*132 :: title
double precision :: znuclpsp,zionpsp
integer :: pspso,pspdat,pspcod,pspxc,lmax,lloc,mmax=integers
double precision :: residm,xred(3,natom),etotal,fermie
```

NOTE: *etotal* is set to its true value only for density and potential files. For other files, it is set to 1.0d20.

NOTE: *ecut_eff* = *ecut* * *dilatmx2*.

NOTE: In pre–v4.1, fermie is set to zero for non–metallic occupation numbers, or for non–self–consistent calculations. In v4.1 and later, for all cases where occupation numbers are defined (that is, positive *iscf*, and *iscf*=−3), and for non–metallic occupation numbers, the Fermi energy is set to the highest occupied eigenenergy.

The header might differ for different versions of ABINIT. The pre–v4.2 formats are described below. Note however, that the current version of ABINIT is able to read all these formats (not to write them).

The format for version 4.1 was:

```
 write(unit=header) codvsn,headform,fform
 write(unit=header) bantot,date,intxc,ixc,natom,ngfft(1:3),&
& nkpt,nspden,nspinor,nsppol,nsym,npsp,ntypat,occopt,pertcase,&
& ecut,ecutsm,ecut_eff,qptn(1:3),rprimd(1:3,1:3),tphysel,tsmear
 write(unit=header) istwfk(1:nkpt),nband(1:nkpt*nsppol),&
& npwarr(1:nkpt),so_typat(1:ntypat),symafm(1:nsym),&
& symrel(1:3,1:3,1:nsym),typat(1:natom),&
& kpt(1:3,1:nkpt),occ(1:bantot),tnons(1:3,1:nsym),znucltypat(1:ntypat)
 do ipsp=1,npsp
! (npsp lines, 1 for each pseudopotential ; npsp=ntypat, except if
! alchemical pseudo-atoms)
  write(unit=unit) title,znuclpsp,zionpsp,pspso,pspdat,pspcod,pspxc
 enddo
!(final record: residm, coordinates, total energy, Fermi energy)
 write(unit=unit) residm,xred(1:3,1:natom),etotal,fermie
```

The format for version 4.0 was:

```
 write(unit=header) codvsn,headform,fform
 write(unit=header) bantot,date,intxc,ixc,natom,ngfft(1:3),&
& nkpt,nspden,nspinor,nsppol,nsym,npsp,ntypat,occopt,&
& ecut,ecutsm,ecut_eff,rprimd(1:3,1:3),tphysel,tsmear
 write(unit=header) istwfk(1:nkpt),nband(1:nkpt*nsppol),&
& npwarr(1:nkpt),so_typat(1:ntypat),symafm(1:nsym),&
& symrel(1:3,1:3,1:nsym),typat(1:natom),&
& kpt(1:3,1:nkpt),occ(1:bantot),tnons(1:3,1:nsym),znucltypat(1:ntypat)
 do ipsp=1,npsp
! (npsp lines, 1 for each pseudopotential ; npsp=ntypat, except if
! alchemical pseudo-atoms)
  write(unit=unit) title,znuclpsp,zionpsp,pspso,pspdat,pspcod,pspxc
 enddo
!(final record: residm, coordinates, total energy, Fermi energy)
 write(unit=unit) residm,xred(1:3,1:natom),etotal,fermie
```

The format for version 3.4 was:

```
 write(unit=header) codvsn,headform,fform
 write(unit=header) bantot,date,intxc,ixc,natom,ngfft(1:3),&
& nkpt,nspden,nspinor,nsppol,nsym,npsp,ntypat,occopt,ecut_eff,rprimd(1:3,1:3)
 write(unit=header) nband(1:nkpt*nsppol),&
& npwarr(1:nkpt),symrel(1:3,1:3,1:nsym),typat(1:natom),istwfk(1:nkpt),&
& kpt(1:3,1:nkpt),occ(1:bantot),tnons(1:3,1:nsym),znucltypat(1:ntypat)
 do ipsp=1,npsp
! (npsp lines, 1 for each pseudopotential ; npsp=ntypat, except if
! alchemical pseudo-atoms)
  write(unit=unit) title,znuclpsp,zionpsp,pspso,pspdat,pspcod,pspxc
 enddo
!(final record: residm, coordinates, total energy, Fermi energy)
 write(unit=unit) residm,xred(1:3,1:natom),etotal,fermie
```

The format for versions 2.3 to 3.3 was:

```
 write(unit=header) codvsn,headform,fform
 write(unit=header) bantot,date,intxc,ixc,natom,ngfft(1:3),&
& nkpt,nspden,nspinor,nsppol,nsym,ntypat,occopt,acell(1:3),&
& ecut_eff,rprimd(1:3,1:3)
 write(unit=header) nband(1:nkpt*nsppol),&
& npwarr(1:nkpt),symrel(1:3,1:3,1:nsym),typat(1:natom),istwfk(1:nkpt),&
& kpt(1:3,1:nkpt),occ(1:bantot),tnons(1:3,1:nsym),znucl(1:ntypat)
 do itypat=1,ntypat
! (ntypat lines, 1 for each psp...)
  write(unit=unit) title,znucl,zion,pspso,pspdat,pspcod,pspxc,&
& lmax,lloc,mmax
 enddo
!(final record: residm, coordinates, total energy, Fermi energy)
 write(unit=unit) residm,xred(1:3,1:natom),etotal,fermie
```

The format for versions 2.0, 2.1 and 2.2 was:

```
 write(unit=header) codvsn,fform
 write(unit=header) bantot,date,intxc,ixc,natom,ngfft(1:3),&
& nkpt,nsppol,nsym,ntypat,acell(1:3),ecut_eff,rprimd(1:3,1:3)
 write(unit=header) nband(1:nkpt*nsppol),&
& npwarr(1:nkpt),symrel(1:3,1:3,1:nsym),typat(1:natom),istwfk(1:nkpt),&
& kpt(1:3,1:nkpt),occ(1:bantot),tnons(1:3,1:nsym),znucl(1:ntypat)
 do itypat=1,ntypat
! (ntypat lines, 1 for each psp...)
  write(unit=unit)   title,znucl,zion,pspdat,pspcod,pspxc,lmax,lloc,mmax
 enddo
!(final record: residm, coordinates, total energy)
 write(unit=unit) residm,xred(1:3,1:natom),etotal
```

6.5.5 The density output file

This is an unformatted data file containing the electron density on the real space FFT grid. It consists of the header records followed by

```
do ispden=1,nspden
 write(unit) (rhor(ir),ir=1,cplex*ngfft(1)*ngfft(2)*ngfft(3))
enddo
```

where **rhor** is the electron density in electrons/bohr^3, and cplex is the number of complex components of the density (cplex=1 for GS calculations — the density is real —, and cplex=1 or 2 for RF). The input variable *nspden* describes the number of components of the density. The first component (the only one present when *nspden*=1) is always the total charge density. When *nspden*=2, the second component is the density associated with spin–up electrons. The case *nspden*=4 is not yet implemented. Note that the meaning of the different components of the density differs for the density array (rhor) and for the different potential arrays (vxc ...), see section 6.6.

To identify the points in real space which correspond with the index "ir" above, consider the following.

The first array value (ir=1) corresponds with the first grid point which is at the origin of the unit cell, ($x = 0$, $y = 0$, $z = 0$).

The next grid point (ir=2) lies along the first primitive translation at the next fft grid point, which is (1/*ngfft*(1))**acell*(1)**rprim*(mu,1). This is 1/*ngfft*(1) of the way along the first primitive translation.

The rest of the values up to ir=*ngfft*(1) lie along this vector, at (ir-1)/*ngfft*(1) of the way along the first primitive translation. The point at ir=*ngfft*(1)+1 lies at 1/*ngfft*(2) along the second primitive translation.

The next points up to ir=*ngfft*(1)+*ngfft*(1) are displaced in the direction of the second primitive translation by 1/*ngfft*(2) and in the first translation by (ir-*ngfft*(1)-1)/*ngfft*(1).

This pattern continues until ir=*ngfft*(1)*ngfft*(2).

The next point after that is displaced along the third primitive translation by 1/*ngfft*(3), and so forth until ir varies all the way from 1 to *ngfft*(1)*ngfft*(2)*ngfft*(3). This last point is in the corner diagonally opposite from the origin, or right alongside the origin if the whole grid is viewed as being periodically repeated.

6.5.6 The potential files

Also unformatted files consisting of the header records and

```
do ispden=1,nspden
 write(unit) (potential(ir),ir=1,cplex*ngfft(1)*ngfft(2)*ngfft(3))
enddo
```

where **potential** can be either the sum of the Hartree potential, exchange–correlation and local pseudopotential (see *prtopt*), the Hartree potential (see *prtvha*), the Hartree+XC potential (see *prtvhxc*), or the XC potential (see *prtvxc*), These are defined on the real space grid in Hartree energy units. The underlying grid is as described above. If *nspden*=2, the different components are the spin–up potential and the spin–down potential. The case *nspden*=4 is not yet implemented. Note that the Hartree potential is NOT spin–dependent, but in order to use the same format as for the other potential files, the spin–independent array is written twice, once for spin–up and one for spin–down.

6.5.7 The wavefunction output file

This is an unformatted data file containing the planewaves coefficients of all the wavefunctions, and different supplementary data.

The **ground–state** wf file consists of the header records, and data written with the following lines of FORTRAN (version 4.0 and more recent versions):

```
bantot=0                                        <-- counts over all bands
do isppol=1,nsppol
 do ikpt=1,nkpt
  write(unit) npw,nspinor,nband                    <-- for each $k$--point
  write(unit) kg(1:3,1:npw)                        <-- plane wave reduced coordinates
  write(unit) eigen(1+bantot:nband+bantot),        <-- eigenvalues for this $k$--point
              occ(1+bantot:nband+bantot)           <-- occupation numbers for this k point
  do iband=1,nband
   write(unit) (cg(ii+...),ii=1,2*npw*nspinor) <-- wavefunction coefficients
  enddo                                            for a single band and k point
  bantot=bantot+nband
 enddo
enddo
```

If the job ended without problem, and if one is not using **newsp**, a few supplementary lines are added, in order to give the history of atomic positions and corresponding forces. The integer nxfh gives the number of pairs (x,f) of positions and forces in reduced coordinates:

```
 write(unit)nxfh
 do ixfh=1,nxfh
  write(unit) xred(1:3,1:natom,ixfh),dummy(1:3,1:4),&
```

```
&             fred(1:3,1:natom,ixfh),dummy(1:3,1:4)
 enddo
```

The dummy variables might contain, in the future, the description of the unit cell, and the stresses. The type of the different variables is:

```
integer :: kg,nband,npw,nspinor,nxfh
double precision :: cg,dummy,eigen,fred,occ,xred
```

The **response–function** wf file consists of the header records, and data written with the following lines of FORTRAN (version 4.0 and more recent versions):

```
bantot=0                                        <-- counts over all bands
do isppol=1,nsppol
 do ikpt=1,nkpt
  write(unit) npw,nspinor,nband                     <-- for each k point
  write(unit) kg(1:3,1:npw)                         <-- plane wave reduced coordinates
  do iband=1,nband
   write(unit) (eigen(jband+(iband-1)*nband+bantot),jband=1,2*nband)
                                                    <-- column of eigenvalue matrix
   write(unit) (cg(ii+...),ii=1,2*npw*nspinor)      <-- wavefunction coefficients
  enddo                                             for a single band and k point
  bantot=bantot+nband
 enddo
enddo
```

In version previous to 4.0, npw and nspinor were combined:

```
write(unit) npw*nspinor,nband
```

while the planewave coordinate record was not present (in both GS and RF cases).

Note that there is an alternative format (`_KSS`) for the output of the wavefunction coefficients, activated by a non–zero value of *nbandkss*.

6.5.8 Other output files

There are many other output files, optionally written, all formatted files at present. Their use is usually governed by a specific input variable. Please consult the description of this input variable, in order to have more information on such files:

- *prtcml* to print a CML file with geometry information;
- *prtdos* to print a file with the electronic Density–Of–States;
- *prtgeo* to print a file with a geometrical analysis (bond lengths and bond angles), that also contains an XMOL section;
- *prt1dm* to print a one–dimensional projection of potential and density, for the three axes.

If you follow the tutorial, you should go back to the tutorial window now.

6.6 Numerical quality of the calculations

The following section describes various parameters which affect convergence and the numerical quality of calculations.

The list of these input parameters is

1. *ecut*;

2. *toldfe*, *toldff*, *tolwfr*, and *tolvrs*, as well as *nstep*;
3. *nkpt*;
4. *ngfft*;
5. *tolmxf*, as well as *amu*, *dtion*, *vis*, *ntime*;
6. *acell* and *rprim*.

The technical design of the pseudopotential also affects the quality of the results.

1. The first issue regarding convergence is the number of planewaves in the basis for a given set of atoms. Some atoms (notably those in the first row or first transition series row) have relatively deep pseudopotentials which require many planewaves for convergence. In contrast are atoms like Si for which fewer planewaves are needed. A typical value of "ecut" for silicon might be 5–10 Hartree for quite good convergence, while the value for oxygen might be 25–35 hartree or more depending on the convergence desired and the design of the pseudo–potential.

   NOTE: It is necessary in every new problem to **TEST** the convergence by **RAISING** *ecut* for a given calculation until the results being computed are constant to within some tolerance. This is up to the user and is very important. For a given *acell* and *rprim*, *ecut* is the parameter which controls the number of planewaves. Of course if *rprim* or *acell* is varied then the number of planewaves will also change.

   Let us reiterate that extremely careful pseudopotential design can optimize the convergence of e.g. the total energy within some range of planewave number or *ecut*. It is appropriate to attempt to optimize this convergence, especially for difficult atoms like oxygen or copper, as long as one does not significantly compromise the quality or transferability of the pseudopotential. There are many people working on new techniques for optimizing convergence.

   For information on extended norm conservation, see E. L. Shirley, D. C. Allan, R. M. Martin, and J. D. Joannopoulos, Phys. Rev. B 40, 3652 (1989).

   For information on optimizing the convergence of pseudopotentials, see A. M. Rappe, K. M. Rabe, E. Kaxiras, and J. D. Joannopoulos, Phys. Rev. B 41, 1227 (1990).

2. In addition to achieving convergence in the number of planewaves in the basis, one must ensure that the SCF iterations which solve the electronic structure for a given set of atomic coordinates are also converged. This convergence is controlled by the parameters *toldfe*, *toldff*, *tolwfr*, and *tolvrs*, as well as the parameter *nstep*. One of the "tolerance" parameters must be chosen, and, when the required level of tolerance is fulfilled, the SCF cycles will stop. The *nstep* variable also controls convergence in preconditioned conjugate gradient iterations by forcing the calculation to stop whenever the number of such iterations exceeds *nstep*. Usually one wants *nstep* to be set larger than needed to reach a given tolerance, or else one wants to restart insufficiently converged calculations until the required tolerance is reached.

   Note that, if the gap in the system closes (e.g. due to defect formation or if the system is metallic in the first place), the presently coded algorithm will be slower to converge than for insulating materials. Convergence trouble during iterations usually signals closure of the gap. The code will suggest to treat at least one unoccupied state (or band) in order to be able to monitor such a closure.

3. For self consistent calculations (*iscf* positive) it is important to test the adequacy of the k–point integration. If symmetry is used then one usually tests a set of "special point" grids. Otherwise one tests the addition of more and more k–points, presumably on uniform grids, to ensure that a sufficient number has been included for good k–point integration. The parameter *nkpt* indicates how many k–points are being used, and their coordinates are given by *kpt* and *kptnrm*, described above. The weight given to each k–point is provided

by input variable *wtk*. Systematic tests of k–point integration are much more difficult than tests of the adequacy of the number of planewaves. The difficulty I refer to is simply the lack of a very systematic method for generating k–point grids for tests.

4. It is possible to run calculations for which the fft box is not quite large enough to avoid aliasing error in fft convolutions. An aliasing error, or a fourier filter approximation, is occurring when the output variable "**boxcut**" is less than 2. boxcut is the smallest ratio of the fft box side to the planewave basis sphere diameter. If this ratio is 2 or larger then e.g. the calculation of the Hartree potential from the charge density is done without approximation.

   NOTE: the values of *ngfft*(1:3) are chosen automatically by the code to give boxcut ¿ 2, if *ngfft* has not been set by hand. At ratios smaller than 2, certain of the highest fourier components are corrupted in the convolution. If the basis is nearly complete, this fourier filter can be an excellent approximation. In this case values of boxcut can be as small as about 1.5 without incurring significant error. For a given *ecut*, *acell*, and *rprim*, one should run tests for which ngfft is large enough to give boxcut ≥ 2, and then one may try smaller values of *ngfft* if the results are not significantly altered. See the descriptions of these variables above.

5. If you are running calculations to relax or equilibrate structures, i.e. with *ionmov*=1 and possibly *vis*¿0, then the quality of your molecular dynamics or relaxation will be affected by the parameters *amu*, *dtion*, *vis*, *ntime*, *tolmxf*. Clearly if you want a relaxed structure you must either run long enough or make repeated runs until the largest force in the problem (output as fmax) is smaller than what you will tolerate (see *tolmxf*). If dtion is too large for the given values of masses (*amu*) and viscosity (*vis*) then the molecular dynamics will be unstable. If *dtion* is too small, then the molecular dynamics will move inefficiently slowly. A consensus exists in the community that forces larger than about 0.1 eV/Angstrom are really too large to consider the relaxation to be converged. It is best for the user to get experience with this in his/her own application. The option *ionmov*=2, 3 or 7 are also available This uses the Broyden (BFGS) scheme for structural optimization and is much more efficient than viscous damping for structural relaxation.

6. If you are running supercell calculations (i.e. an isolated atom or molecule in a big box, or a defect in a solid, or a slab calculation) you must check the convergence of your calculation with respect to the supercell and system size.

   - For an isolated molecule in a big box: increase concurrently the three dimensions of your supercell (*acell*), and check the convergence of your physical property.
   - For a defect in a solid: your supercell must bu a multiple of the primitive cell of the bulk solid, so you have less freedom. Still, be sure that your supercell is large enough for your properties of interest to be accurate at the level you want it to be.
   - For a slab calculation: you must increase the vacuum in the cell, but also the thickness of your slab systematically ...

If you follow the tutorial, you should go back to the tutorial window now.

6.7 Final remarks

The ABINIT package is developped by the ABINIT group. The status of this package and the ABINIT group are explained in the file `~ABINIT/Infos/context` and `~ABINIT/Infos/planning`, or some recent version of them.

Please send questions and constructive criticisms of the code or this documentation, as well as bug reports (see `~ABINIT/Infos/bug_report`) to

Xavier Gonze

Unité PCPM, Université Catholique de Louvain
1, place Croix du Sud
B–1348 Louvain-la-Neuve
Belgium
tel: (+32) 10 472076
fax: (+32) 10 473452
email: gonze@pcpm.ucl.ac.be

or to

Douglas C. Allan

SP–FR–05
Corning Incorporated
Corning, NY 14831
USA
tel: (+1) 607 974 3498
fax: (+1) 607 974 3675
email: allandc@corning.com

Correspondence by email is usually most convenient.

Chapter 7

Main ABINIT code, input variables: Complete list

This document lists the names (keywords) of all input variables to be used in the main input file of the abinis code.

The new user is advised to read first "the new user's guide" (chapter 4), before reading the present file. It will be easier to discover the present file with the help of the tutorial.

When the user is sufficiently familiarized with ABINIT, the reading of the `~ABINIT/Infos/tuning` file might be useful. For response–function calculations using abinis, the complementary file `~ABINIT/Infos/respfn_help` is needed.

7.1 Basic variables, VARBAS

7.1.1 acell

Mnemonics: scAle CELL
Characteristic: EVOLVING, LENGTH
Variable type: real array acell(3)

Gives the length scales by which dimensionless primitive translations (in *rprim*) are to be multiplied. By default, given in Bohr atomic units (1 Bohr=0.5291772083 Å), although Angstrom can be specified, if preferred, since *acell* has the 'LENGTH' characteristics. See further description of *acell* related to the *rprim* input variable.

7.1.2 angdeg

Mnemonics: ANGles in DEGrees
Characteristic:
Variable type: real array angdeg(3)
No Default (use *rprim* as Default).

Gives the angles between directions of primitive vectors of the unit cell (in degrees), as an alternative to the input array *rprim*. Will be used to set up *rprim*, that, together with the array *acell*, will be used to define the primitive vectors.

- *angdeg*(1) is the angle between the 2nd and 3rd vectors,

- *angdeg*(2) is the angle between the 1st and 3rd vectors,
- *angdeg*(3) is the angle between the 1st and 2nd vectors,

If the three angles are equal within 1.0×10^{-12} (except if they are exactly 90 degrees), the three primitive vectors are chosen so that the trigonal symmetry that exchange them is along the z cartesian axis:

$$R1 = (a, 0, c) R2 = (-a/2, \sqrt{3}/2 * a, c) R3 = (-a/2, -\sqrt{3}/2 * a, c) \quad (7.1)$$

where $a^2 + c^2 = 1.0$.

If the angles are not all equal (or if they are all 90 degrees), one will have the following generic form:

- R1=(1,0,0)
- R2=(a,b,0)
- R3=(c,d,e)

where each of the vectors is normalized, and form the desired angles with the others.

7.1.3 ecut

Mnemonics: Energy CUToff
Characteristic: ENERGY
Variable type: real parameter

Used for kinetic energy cutoff which controls number of planewaves at given k–point by: $(1/2)[(2\pi) * (k + G_{max})]^2 = ecut$ for Gmax. All planewaves inside this "basis sphere" centered at k are included in the basis (except if *dilatmx* is defined). Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV) This is the single parameter which can have an enormous effect on the quality of a calculation; basically the larger *ecut* is, the better converged the calculation is. For fixed geometry, the total energy MUST always decrease as *ecut* is raised because of the variational nature of the problem.

Usually one runs at least several calculations at various ecut *to investigate the convergence needed for reliable results.*

For k–points whose coordinates are build from 0 or 1/2, the implementation of time–reversal symmetry that links coefficients of the wavefunctions in reciprocal space has been realized. See the input variable *istwfk*. If activated (which corresponds to the Default mode), this input variable *istwfk* will allow to divide the number of plane wave (npw) treated explicitly by a factor of two. Still, the final result should be identical with the 'full' set of plane waves.

See the input variable *ecutsm*, for the smoothing of the kinetic energy, needed to optimize unit cell parameters.

7.1.4 iscf

Mnemonics: Integer for Self–Consistent–Field cycles
Characteristic:
Variable type: integer parameter
Default is 5.

Control the self–consistency.

Positive, non–zero values ⇒ this is the usual choice for doing the usual ground state (GS) calculations or for structural relaxation, where the potential has to be determined self–consistently. The choice between different algorithms for SCF is possible:

- =1 ⇒ get the largest eigenvalue of the SCF cycle (DEVELOP option, used with *irdwfk*=1 or *irdwfq*=1);
- =2 ⇒ SCF cycle, simple mixing;
- =3 ⇒ SCF cycle, anderson mixing;
- =5 ⇒ SCF cycle, CG based on the minim. of the energy;
- Other positive, including zero values, are not allowed.

The preferred option is 5, which is quite robust. The value 3 can be faster, but sometimes the SCF iterations will not converge with *iscf*=3! Other (negative) options:

- $= -2$ ⇒ a non–self–consistent calculation is to be done; in this case an electron density rho(r) on a real space grid (produced in a previous calculation) will be read from a disk file (automatically if *ndtset*=0, or according to the value of *getden* if *ndtset* $\neq 0$). The name of the density file must be given as indicated in the section 4 of `abinis_help`. *iscf* $= -2$ would be used for band structure calculations, to permit computation of the eigenvalues of occupied and unoccupied states at arbitrary k–points in the fixed self consistent potential produced by some integration grid of k–points. To compute the eigenvalues (and wavefunctions) of unoccupied states in a separate (non–self–consistent) run, the user should save the self–consistent rho(r) and then run *iscf* $= -2$ for the intended set of k–points and bands. To prepare a run with *iscf* $= -2$, a density file can be produced using the parameter *prtden* (see its description). When a self–consistent set of wavefunctions is already available, abinit can be used with *nstep*=0 (see `Test_v2/t47.in`), and the adequate value of *prtden*.
- $= -3$ ⇒ like -2, but initialize *occ* and *wtk*, directly or indirectly (using *ngkpt* or *kptrlatt*) depending on the value of *occopt*. For GS, this option might be used to generate Density–of–states (thanks to *prtdos*), or to produce STM charge density map (thanks to *prtstm*). For RF, this option is needed to compute the response to ddk perturbation.
- $= -1$ ⇒ like -2, but the non–self–consistent calculation is followed by the determination of excited states within TDDFT. This is only possible for *nkpt*=1, *kpt*=0 0 0, *nsppol*=1. Note that the oscillator strength needs to be defined with respect to an origin of coordinate, thanks to the input variable *boxcenter*. The maximal number of Kohn–Sham excitations to be used to build the excited state TDDFT matrix can be defined by *td_mexcit*, or indirectly by the maximum Kohn–Sham excitation energy *td_maxene*.

7.1.5 ixc

Mnemonics: Integer for eXchange–Correlation choice
Characteristic:
Variable type: integer parameter
Default is *ixc*=1 (Teter parameterization). However, if all the pseudopotentials have the same value of pspxc, the initial value of *ixc* will be that common value.

Control the choice of exchange and correlation (xc).

- 0 ⇒ NO xc;
- 1 ⇒ LDA or LSD, Teter Pade parametrization (4/93, published in S. Goedecker, M. Teter, J. Huetter, Phys. Rev. B 54, 1703 (1996)), which reproduces Perdew–Wang (which reproduces Ceperley–Alder!).
- 2 ⇒ LDA, Perdew–Zunger–Ceperley–Alder (no spin–polarization)
- 3 ⇒ LDA, old Teter rational polynomial parametrization (4/91) fit to Ceperley–Alder data (no spin–polarization)

- 4 ⇒ LDA, Wigner functional (no spin–polarization)
- 5 ⇒ LDA, Hedin–Lundqvist functional (no spin–polarization)
- 6 ⇒ LDA, "X–alpha" functional (no spin–polarization)
- 7 ⇒ LDA or LSD, Perdew–Wang 92 functional
- 8 ⇒ LDA or LSD, x–only part of the Perdew–Wang 92 functional
- 9 ⇒ LDA or LSD, x– and RPA correlation part of the Perdew–Wang 92 functional
- 11 ⇒ GGA, Perdew–Burke–Ernzerhof GGA functional
- 12 ⇒ GGA, x–only part of Perdew–Burke–Ernzerhof GGA functional
- 13 ⇒ GGA potential of van Leeuwen–Baerends, while for energy, Perdew–Wang 92 functional
- 14 ⇒ GGA, revPBE of Y. Zhang and W. Yang, Phys. Rev. Lett. 80, 890 (1998)
- 15 ⇒ GGA, RPBE of B. Hammer, L.B. Hansen and J.K. Norskov, Phys. Rev. B 59, 7413 (1999)
- 16 ⇒ GGA, HTCH of F.A. Hamprecht, A.J. Cohen, D.J. Tozer, N.C. Handy, J. Chem. Phys. 109, 6264 (1998)
- 20 ⇒ Fermi–Amaldi xc ($-1/N$ Hartree energy, where N is the number of electrons per cell; $G = 0$ is not taken into account however), for TDDFT tests. No spin–pol. Does not work for RF.
- 21 ⇒ same as 20, except that the xc–kernel is the LDA (*ixc*=1) one, for TDDFT tests.
- 22 ⇒ same as 20, except that the xc–kernel is the Burke–Petersilka–Gross hybrid, for TDDFT tests.

Note that the choice made here should agree with the choice made in generating the original pseudopotential, except for *ixc*=0 (usually only used for debugging). A warning is issued if this is not the case. However, the choices *ixc*=1, 2, 3 and 7 are fits to the same data, from Ceperley–Alder, and are rather similar, at least for spin–unpolarized systems. The choice between the LDA or the LSD is governed by the value of *nsppol* (see below).

NOTE: in the implementation of the spin–dependence of these functionals, and in order to avoid divergences in their derivatives, the interpolating function between spin–unpolarized and fully–spin–polarized function has been slightly modified, by including a zeta rescaled by 1.d0-1.d-6. This should affect total energy at the level of 1.d-6Ha, and should have an even smaller effect on differences of energies, or derivatives. The value *ixc*=10 is used internally: gives the difference between *ixc*=7 and *ixc*=9, for use with an accurate RPA correlation energy.

7.1.6 jdtset

Mnemonics: index -J- for DaTaSETs
Characteristic: NO MULTI
Variable type: integer array jdtset(*ndtset*)
Default: the series 1, 2, 3 ... *ndtset*.

Gives the dataset index of each of the datasets. This index will be used:

- to determine which input variables are specific to each dataset, since the variable names for this dataset will be made from the bare variable name concatenated with this index, and only if such a composite variable name does not exist, the code will consider the bare variable name, or even, the Default;

- to characterize output variable names, if their content differs from dataset to dataset;
- to characterize output files (root names appended with _DSx where 'x' is the dataset index).

The allowed index values are between 1 and 99.

An input variable name appended with 0 is not allowed.

When *ndtset*==0, this array is not used, and moreover, no input variable name appended with a digit is allowed. This array might be initialized thanks to the use of the input variable *udtset*. In this case, *jdtset* cannot be used.

7.1.7 kpt

Mnemonics: K - PoinTs
Characteristic:
Variable type: real array kpt(3,*nkpt*)
Default is 0. 0. 0. (for just one k–point)

Contains the k–points in terms of reciprocal space primitive translations (NOT in cartesian coordinates!). Needed ONLY if *kptopt*=0, otherwise deduced from other input variables.

It contains dimensionless numbers in terms of which the cartesian coordinates would be: k_cartesian = k1*G1+k2*G2+k3*G3 where (k1, k2, k3) represent the dimensionless "reduced coordinates" and G1, G2, G3 are the cartesian coordinates of the primitive translation vectors. G1, G2, G3 are related to the choice of direct space primitive translation vectors made in *rprim*. Note that an overall norm for the k–points is supplied by *kptnrm*. This allows one to avoid supplying many digits for the k–points to represent such points as (1,1,1)/3.

Note: one of the algorithms used to set up the sphere of G vectors for the basis needs components of k–points in the range $[-1, 1]$, so the remapping is easily done by adding or subtracting 1 from each component until it is in the range $[-1, 1]$. That is, given the k–point normalization *kptnrm* described below, each component must lie in [–*kptnrm*, *kptnrm*]. Not read if *kptopt* $\neq 0$.

7.1.8 kptnrm

Mnemonics: K - PoinTs NoRMalization
Characteristic:
Variable type: real parameter
Default is 1.

Establishes a normalizing denominator for each k–point. Needed only if *kptopt* ≤ 0, otherwise deduced from other input variables. The k–point coordinates as fractions of reciprocal lattice translations are therefore *kpt*(mu,ikpt)/*kptnrm*. *kptnrm* defaults to 1 and can be ignored by the user. It is introduced to avoid the need for many digits in representing numbers such as 1/3.

It cannot be smaller than 1.0.

7.1.9 kptopt

Mnemonics: KPoinTs OPTion
Characteristic:
Variable type: integer parameter
Default is 0.

Control the set up of the k–points list. The aim will be to initialize, by straight reading or by a preprocessing approach based on other input variables, the following input variables, giving the k–points, their number, and their weight: *kpt*, *kptnrm*, *nkpt*, and, for *iscf* $\neq -2$, *wtk*.

Often, the k–points will form a lattice in reciprocal space. In this case, one will also aim at initializing input variables that give the reciprocal of this k–point lattice, as well as its shift with respect to the origin: *ngkpt* or *kptrlatt*, as well as on *nshiftk* and *shiftk*.

- 0 ⇒ read directly *nkpt*, *kpt*, *kptnrm* and *wtk* (corresponds to the usage before version 2.1)

  One can use the kptgen utility to produce these input data.

- 1 ⇒ rely on *ngkpt* or *kptrlatt*, as well as on *nshiftk* and *shiftk* to set up the k–points. Take fully into account the symmetry to generate the k–points in the Irreducible Brillouin Zone only.

  (This is the usual mode for GS calculations)

- 2 ⇒ rely on *ngkpt* or *kptrlatt*, as well as on *nshiftk* and *shiftk* to set up the k–points. Take into account only the time–reversal symmetry: k–points will be generated in half the Brillouin zone.

  (This is to be used when preparing or executing a RF calculation at q=(0 0 0))

- 3 ⇒ rely on *ngkpt* or *kptrlatt*, as well as on *nshiftk* and *shiftk* to set up the k–points. Do not take into account any symmetry: k–points will be generated in the full Brillouin zone.

  (This is to be used when preparing or executing a RF calculation at non–zero q)

- (4 ⇒ has been replaced by negative values in version 2.3)

- A negative value ⇒ rely on *kptbounds*, and *ndivk* to set up a band structure calculation along different lines (allowed only for *iscf*==−2). The absolute value of *kptopt* gives the number of segments of the band structure.

In the case of a grid of k–points, the auxiliary variables *kptrlen*, *ngkpt* and *prtkpt* might help you to select the optimal grid.

7.1.10 natom

Mnemonics: Number of ATOMs
Characteristic:
Variable type: integer parameter
Default is 1

Gives the total number of atoms in the unit cell. Default is 1 but you will obviously want to input this value explicitly. Note that *natom* refers to all atoms in the unit cell, not only to the irreducible set of atoms in the unit cell (using symmetry operations, this set allows to recover all atoms). If you want to specify only the irreducible set of atoms, use the symmetriser, see the input variable *natrd*.

7.1.11 nband

Mnemonics: Number of BANDs
Characteristic:
Variable type: integer parameter
Default is 1.

Gives number of bands, occupied plus possibly unoccupied, for which wavefunctions are being computed along with eigenvalues. Note: if the parameter *occopt* (see below) is not set to 2, *nband* is a scalar integer, but if the parameter *occopt* is set to 2, then *nband* must be an array *nband*(*nkpt* * *nsppol*) giving the number of bands explicitly for each k–point. This option is provided in order to allow the number of bands treated to vary from k–point to k–point. For the values of *occopt*

not equal to 0 or 2, *nband* can be omitted. The number of bands will be set up thanks to the use of the variable *fband*. The present Default will not be used.

If *nspinor* is 2, *nband* must be even for each k–point.

In the case of a GW calculation (*optdriver*=3 or 4), *nband* gives the number of bands to be treated to generate the screening (susceptibility and dielectric matrix), as well as the self–energy. However, to generate the _KSS file (see *kssform*) the relevant number of bands is given by *nbandkss*.

7.1.12 ndtset

Mnemonics: Number of DaTaSETs
Characteristic: NO MULTI
Variable type: integer parameter
Default is 0 (no multi–data set).

Gives the number of data sets to be treated.

If 0, means that the multi–data set treatment is not used, so that the root filenames will not be appended with _DSx, where 'x' is the dataset index defined by the input variable *jdtset*, and also that input names with a dataset index are not allowed. Otherwise, *ndtset*=0 is equivalent to *ndtset*=1.

7.1.13 ngkpt

Mnemonics: Number of Grid points for K Points generation
Characteristic: NOT INTERNAL
Variable type: integer array ngkpt(3)
No Default

Used when *kptopt* ≥ 0, if *kptrlatt* has not been defined (*kptrlatt* and *ngkpt* are exclusive of each other). Its three positive components give the number of k–points of Monkhorst–Pack grids (defined with respect to primitive axis in reciprocal space) in each of the three dimensions. *ngkpt* will be used to generate the corresponding *kptrlatt* input variable. The use of *nshiftk* and *shiftk*, allows to generate shifted grids, or Monkhorst–Pack grids defined with respect to conventional unit cells.

When *nshiftk*=1, *kptrlatt* is initialized as a diagonal (3x3) matrix, whose diagonal elements are the three values *ngkpt*(1:3). When *nshiftk* is greater than 1, ABINIT will try to generate *kptrlatt* on the basis of the primitive vectors of the k–lattice: the number of shifts might be reduced, in which case *kptrlatt* will not be diagonal anymore.

Monkhorst–Pack grids are usually the most efficient when their defining integer numbers are even. For a measure of the efficiency, see the input variable *kptrlen*.

7.1.14 nkpt

Mnemonics: Number of K - Points
Characteristic:
Variable type: integer parameter
Default is 0 if *kptopt*$\neq$0, and 1 if *kptopt*==0.

If non–zero, *nkpt* gives the number of k–points in the k–point array *kpt*. These points are used either to sample the Brillouin zone, or to build a band structure along specified lines.

If *nkpt* is zero, the code deduces from other input variables (see the list in the description of *kptopt*) the number of k–points, which is possible only when *kptopt*$\neq$0. If *kptopt*$\neq$0 and the input value of *nkpt*$\neq$0, then ABINIT will check that the number of k–points generated from the other input variables is exactly the same than *nkpt*.

If *kptopt* is positive, *nkpt* must be coherent with the values of *kptrlatt*, *nshiftk* and *shiftk*. For ground state calculations, one should select the k–point in the irreducible Brillouin Zone (obtained by taking into account point symmetries and the time–reversal symmetry). For response function calculations, one should select k–points in the full Brillouin zone, if the wavevector of the perturbation does not vanish, or in a half of the Brillouin Zone if $q = 0$. The code will automatically decrease the number of k–points to the minimal set needed for each particular perturbation.

If *kptopt* is negative, *nkpt* will be the sum of the number of points on the different lines of the band structure. For example, if *kptopt*=−3, one will have three segments; supposing *ndivk* is 10 12 17, the total number of k–points of the circuit will be 10+12+17+1(for the final point)=40.

7.1.15 nshiftk

Mnemonics: Number of SHIFTs for k–point grids
Characteristic:
Variable type: integer parameter
The Default is 1.

This parameter gives the number of shifted grids to be used concurrently to generate the full grid of k–points. It can be used with primitive grids defined either from *ngkpt* or *kptrlatt*. The maximum allowed value of *nshiftk* is 8. The values of the shifts are given by *shiftk*.

7.1.16 nsppol

Mnemonics: Number of SPin POLarization
Characteristic:
Variable type: integer parameter
The Default is 1.

Give the number of independent spin polarisations. Can take the values 1 or 2.

If *nsppol*=1, one has an unpolarized calculation (*nspinor*=1, *nspden*=1) or an antiferromagnetic system (*nspinor*=1, *nspden*=2), or a calculation in which spin up and spin down cannot be disantengled (*nspinor*=2), non–collinear magnetism or presence of spin–orbit coupling, for which one needs spinor wavefunctions.

If *nsppol*=2, one has a spin–polarized calculation with separate and different wavefunctions for up and down spin electrons for each band and k–point. Compatible only with *nspinor*=1, *nspden*=2.

In the present status of development, with *nsppol*=1, all values of *ixc* are allowed, while with *nsppol*=2, only *ixc*=0, 1, 7 and 11 are allowed.

See also the input variable *nspden* for the components of the density matrix with respect to the spin–polarization.

7.1.17 nstep

Mnemonics: Number of self–consistent field STEPS
Characteristic:
Variable type: integer parameter
Default is 1.

Gives the maximum number of SCF cycles (or "iterations"). Full convergence from random numbers if usually achieved in 12-20 SCF iterations. Each can take from minutes to hours. In certain difficult cases, usually related to a small or zero bandgap, convergence performance may be much worse. When the convergence tolerance *tolwfr* on the wavefunctions is satisfied, iterations will stop, so for well converged calculations you should set *nstep* to a value larger than you think will be needed for full convergence, e.g. if 20 steps usually converges the system, set *nstep* to 30.

NOTE that a choice of *nstep*=0 is permitted; this will either read wavefunctions from disk (with *irdwfk*=1 or *irdwfq*=1, or non–zero *getwfk* or *getwfq* in the case of multi–dataset) and compute the density, the total energy and stop, or else (with all of the above vanishing) will initialize randomly the wavefunctions and compute the resulting density and total energy. This is provided for testing purposes. One can output the density by using *prtden*. Unlike the forces, the stress tensor also gets computed with *nstep*=0.

7.1.18 nsym

Mnemonics: Number of SYMmetry operations
Characteristic: SYMMETRY FINDER
Variable type: integer parameter
Default is 0.

Gives number of space group symmetries to be applied in this problem. Symmetries will be input in array "*symrel*" and (nonsymmorphic) translations vectors will be input in array "*tnons*". If there is no symmetry in the problem then set *nsym* to 1, because the identity is still a symmetry. In case of a RF calculation, the code is able to use the symmetries of the system to decrease the number of perturbations to be calculated, and to decrease of the number of special k–points to be used for the sampling of the Brillouin zone. After the response to the perturbations have been calculated, the symmetries are used to generate as many as possible elements of the 2DTE from those already computed.

SYMMETRY FINDER mode (Default mode). If *nsym* is 0, all the atomic coordinates must be explicitely given (one cannot use the geometry builder and the symmetrizer): the code will then find automatically the symmetry operations that leave the lattice and each atomic sublattice invariant. It also checks whether the cell is primitive (see *chkprim*). Note that the tolerance on symmetric atomic positions and lattice is rather stringent: for a symmetry operation to be admitted, the lattice and atomic positions must map on themselves within 1.0e-8.

The user is allowed to set up systems with non–primitive unit cells (i.e. conventional FCC or BCC cells, or supercells without any distortion). In this case, pure translations will be identified as symmetries of the system by the symmetry finder. Then, the combined "pure translation + usual rotation and inversion" symmetry operations can be very numerous. For example, a conventional FCC cell has 192 symmetry operations, instead of the 48 ones of the primitive cell. A maximum limit of 384 symmetry operations is hard–coded. This corresponds to the maximum number of symmetry operations of a 2x2x2 undistorted supercell. Going beyond that number will make the code stop very rapidly. If you want nevertheless, for testing purposes, to treat a larger number of symmetries, change the initialization of "msym" in the abinit.f main routine, then recompile the code.

7.1.19 ntypat

Mnemonics: Number of TYPEs of atoms
Characteristic: NO MULTI
Variable type: integer parameter
Default is 1.

Gives the number of types of atoms. E.g. for a homopolar system (e.g. pure Si) *ntypat* is 1, while for BaTiO3, *ntypat* is 3. Except when alchemical mixing of pseudopotentials is used, the number of types of atoms will be equal to the number of pseudopotentials *npsp* to be provided by the user. Thus, The code will try to read the same number of pseudopotential files, whose names should have been given in the "files" file.

The first pseudopotential will be assigned the type number 1, and so on ...

7.1.20 occopt

Mnemonics: OCCupation OPTion
Characteristic:
Variable type: integer option parameter
The Default is occopt=1.

Control how input parameters *nband*, *occ*, and *wtk* are handled.

- *occopt*=0:
  All k–points have the same number of bands and the same occupancies of bands. *nband* is given as a single number, and *occ*(*nband*) is an array of *nband* elements, read in by the code. The k–point weights in array *wtk*(*nkpt*) are automatically normalized by the code to add to 1.

- *occopt*=1:
  Same as *occopt*=0, except that the array *occ* is automatically generated by the code, to give a semiconductor. An error occurs when filling cannot be done with occupation numbers equal to 2 or 0 in each k–point (non–spin–polarized case), or with occupation numbers equal to 1 or 0 in each k–point (spin–polarized case).

- *occopt*=2:
  k–points may optionally have different numbers of bands and different occupancies. *nband*(*nkpt***nsppol*) is given explicitly as an array of *nkpt***nsppol* elements. *occ*() is given explicitly for all bands at each k–point, and eventually for each spin — the total number of elements is the sum of *nband*(ikpt) over all k–points and spins. The k–point weights *wtk* (*nkpt*) are NOT automatically normalized under this option. *occopt*=3, 4, 5, 6 and 7 Metallic occupation of levels, using different occupation schemes (see below). The corresponding thermal broadening, or cold smearing, is defined by the input variable *tsmear* (see below: the variable xx is the energy in Ha, divided by *tsmear*) Like for *occopt*=1, the variable *occ* is not read All k–points have the same number of bands, *nband* is given as a single number, read by the code. The k–point weights in array *wtk*(*nkpt*) are automatically normalized by the code to add to 1.

  - *occopt*=3:
    Fermi–Dirac smearing (finite–temperature metal) Smeared delta function: 0.25d0/(cosh(xx/2.0d0)**2)
  - *occopt*=4:
    "Cold smearing" of N. Marzari (see his thesis work), with a=-.5634 (minimization of the bump) Smeared delta function: exp(-xx2)/sqrt(pi) * (1.5d0+xx*(-a*1.5d0+xx*(-1.0d0+a*xx)))
  - *occopt*=5:
    "Cold smearing" of N. Marzari (see his thesis work), with a=-.8165 (monotonic function in the tail) Same smeared delta function as *occopt*=4, with different a.
  - *occopt*=6:
    Smearing of Methfessel and Paxton (PRB40,3616(1989)) with Hermite polynomial of degree 2, corresponding to "Cold smearing" of N. Marzari with a=0 (so, same smeared delta function as *occopt*=4, with different a).
  - *occopt*=7:
    Gaussian smearing, corresponding to the 0 order Hermite polynomial of Methfessel and Paxton. Smeared delta function: 1.0d0*exp(-xx**2)/sqrt(pi)

WARNING: one can use metallic occupation of levels in the case of a molecule, in order to avoid any problem with degenerate levels. However, it is adviced NOT to use *occopt*=6 (and to a lesser extent *occopt*=4 and 5), since the associated number of electron versus the Fermi energy is NOT garanteed to be a monotonic function. For true metals, AND a sufficiently dense sampling

of the Brillouin zone, this should not happen, but be cautious! As an indication of this problem, a small variation of input parameters might lead to a jump of total energy, because there might be two or even three possible values of the Fermi energy, and the bissection algorithm find one or the other.

7.1.21 rprim

Mnemonics: Real space PRIMitive translations
Characteristic: EVOLVING (if *ionmov*==2 and *optcell* $\neq 0$)
Variable type: real array rprim(3,3)
Default: 3x3 unity matrix.

Give, in columnwise entry, the three dimensionless primitive translations in real space. If the Default is used, that is, *rprim* is the the unity matrix, the three dimensionless primitive vectors are three unit vectors in cartesian coordinates. Each will be multiplied by the corresponding *acell* value to give the dimensional primitive vectors, called *rprim*d. In the general case, the dimensional cartesian coordinates of the crystal primitive translations R1p, R2p and R3p, see *rprimd*, are

```
R1p(i)=rprim(i,1)*acell(1) for i=1,2,3 (x,y,and z)
R2p(i)=rprim(i,2)*acell(2) for i=1,2,3
R3p(i)=rprim(i,3)*acell(3) for i=1,2,3.
```

The *rprim* variable is thus used to define directions of the primitive vectors, that will be multiplied by the appropriate length scale *acell*(1), *acell*(2), or *acell*(3) respectively to give the dimensional primitive translations in real space in cartesian coordinates. Presently, it is requested that the mixed product (R1xR2).R3 is positive. If this is not the case, simply exchange a pair of vectors. To be more specific, *rprim* 1 2 3 4 5 6 7 8 9 corresponds to input of the three primitive translations R1=(1,2,3), R2=(4,5,6), and R3=(7,8,9). Note carefully that the first three numbers input are the first column of *rprim*, the next three are the second, and the final three are the third. This corresponds with the usual Fortran order for arrays. The matrix whose columns are the reciprocal space primitive translations is the inverse transpose of the matrix whose columns are the direct space primitive translations.

Alternatively to *rprim*, directions of dimensionless primitive vectors can be specified by using the input variable *angdeg*. This is especially useful for hexagonal lattices (with 120 or 60 degrees angles). Indeed, in order for symmetries to be recognized, *rprim* must be symmetric up to 10 digits, inducing a specification such as

```
rprim  0.86602540378  0.5  0.0
      -0.86602540378  0.5  0.0
       0.0            0.0  1.0
```

that can be avoided thanks to *angdeg*:

```
angdeg 90 90 120
```

7.1.22 rprimd

Mnemonics: Real space PRIMitive translations, Dimensional
Characteristic: INTERNAL, EVOLVING (if *ionmov*==2 and *optcell* $\neq 0$)
Variable type: real array rprimd(3,3)

This internal variable gives the dimensional real space primitive vectors, computed from *acell* and *rprim*.

```
R1p(i)=rprimd(i,1)=rprim(i,1)*acell(1) for i=1,2,3 (x,y,and z)
R2p(i)=rprimd(i,2)=rprim(i,2)*acell(2) for i=1,2,3
R3p(i)=rprimd(i,3)=rprim(i,3)*acell(3) for i=1,2,3
```

7.1.23 shiftk

Mnemonics: SHIFT for K points
Characteristic:
Variable type: real array shift(3,*nshiftk*)
Default 0.5 0.5 0.5 ... 0.5

It is used only when *kptopt* ≥ 0, and must be defined if *nshiftk* is larger than 1. *shiftk*(1:3,1:*nshiftk*) defines *nshiftk* shifts of the homogeneous grid of k–points based on *ngkpt* or *kptrlatt*. The shifts induced by *shiftk* corresponds to the reduced coordinates in the coordinate system defining the k–point lattice. For example, if the k–point lattice is defined using *ngkpt*, the point whose reciprocal space reduced coordinates are (*shiftk*(1,ii)/*ngkpt*(1) *shiftk*(2,ii)/*ngkpt*(2) *shiftk*(3,ii)/*ngkpt*(3)) belongs to the shifted grid number ii.

The user might rely on ABINIT to suggest suitable and efficients combinations of *kptrlatt* and *shiftk*. The procedure to be followed is described with the input variables *kptrlen*. In what follows, we suggest some interesting values of the shifts, to be used with even values of *ngkpt*. This list is much less exhaustive than the above–mentioned automatic procedure.

1. When the primitive vectors of the lattice do NOT form a FCC or a BCC lattice, the usual (shifted) Monkhorst–Pack grids are formed by using *nshiftk*=1 and *shiftk* 0.5 0.5 0.5. This is often the preferred k–point sampling. For a non–shifted Monkhorst–Pack grid, use *nshiftk*=1 and *shiftk* 0.0 0.0 0.0, but there is little reason to do that.

   The FCC k–point sampling defined with *nshiftk*=4 and *shiftk*

   ```
   0.5 0.5 0.5
   0.5 0.0 0.0
   0.0 0.5 0.0
   0.0 0.0 0.5
   ```

   is particularly efficient.

2. When the primitive vectors of the lattice form a FCC lattice, with *rprim*

   ```
   0.0 0.5 0.5
   0.5 0.0 0.5
   0.5 0.5 0.0
   ```

   the usual Monkhorst–Pack sampling will be generated by using *nshiftk* = 4 and *shiftk*

   ```
   0.5 0.5 0.5
   0.5 0.0 0.0
   0.0 0.5 0.0
   0.0 0.0 0.5
   ```

3. When the primitive vectors of the lattice form a BCC lattice, with *rprim*

   ```
   -0.5  0.5  0.5
    0.5 -0.5  0.5
    0.5  0.5 -0.5
   ```

   the usual Monkhorst–Pack sampling will be generated by using *nshiftk*= 2 and *shiftk*

   ```
    0.25  0.25  0.25
   -0.25 -0.25 -0.25
   ```

   However, the simple sampling *nshiftk*=1 and *shiftk* 0.5 0.5 0.5 is excellent.

4. For hexagonal lattices, one can use *nshiftk*= 1 and *shiftk* 0.0 0.0 0.5.

7.1.24 symrel

Mnemonics: SYMmetry in REaL space
Characteristic:
Variable type: integer array symrel(3,3,*nsym*)
Default is the identity matrix for one symmetry.

Gives "*nsym*" 3x3 matrices expressing space group symmetries in terms of their action on the direct (or real) space primitive translations.

It turns out that these can always be expressed as integers.

Always give the identity matrix even if no other symmetries hold, e.g. *symrel* 1 0 0 0 1 0 0 0 1

Also note that for this array as for all others the array elements are filled in a columnwise order as is usual for Fortran.

The relation between the above symmetry matrices *symrel*, expressed in the basis of primitive translations, and the same symmetry matrices expressed in cartesian coordinates, is as follows. Denote the matrix whose columns are the primitive translations as R, and denote the cartesian symmetry matrix as S. Then

symrel = R(inverse) * S * R

where matrix multiplication is implied. When the symmetry finder is used (see *nsym*), *symrel* will be computed automatically.

7.1.25 tnons

Mnemonics: Translation NON–Symmorphic vectors
Characteristic:
Variable type: real array tnons(3,*nsym*)

Gives the (nonsymmorphic) translation vectors associated with the symmetries expressed in "*symrel*".

These may all be 0, or may be fractional (nonprimitive) translations expressed relative to the real space primitive translations (so, using the "reduced" system of coordinates, see "*xred*"). If all elements of the space group leave 0 0 0 invariant, then these are all 0.

When the symmetry finder is used (see *nsym*), *tnons* is computed automatically.

7.1.26 toldfe

Mnemonics: TOLerance on the DiFference of total Energy
Characteristic:
Variable type: real parameter
Default is 0.0 (stopping condition ignored)

Sets a tolerance for absolute differences of total energy that, reached TWICE successively, will cause one SCF cycle to stop (and ions to be moved).

Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV)

If set to zero, this stopping condition is ignored.

Effective only when SCF cycles are done (*iscf* > 0). In this case, since *toldfe*, *toldff*, *tolvrs* and *tolwfr* are aimed at the same goal (causing the SCF cycle to stop), one and only one of these must be specified.

Because of machine precision, it is not worth to try to obtain differences in energy that are smaller than about 1.0×10^{-12} of the total energy. To get accurate stresses may be quite demanding.

7.1.27 toldff

Mnemonics: TOLerance on the DiFference of Forces
Characteristic:
Variable type: real parameter
Default is 0.0 (stopping condition ignored)

Sets a tolerance for differences of forces (in hartree/bohr) that, reached TWICE successively, will cause one SCF cycle to stop (and ions to be moved).

If set to zero, this stopping condition is ignored.

Effective only when SCF cycles are done (*iscf* > 0). In this case, since *toldfe*, *toldff*, *tolvrs* and *tolwfr* are aimed at the same goal (causing the SCF cycle to stop), one and only one of these must be specified. This tolerance applies to any particular cartesian component of any atom, INCLUDING fixed ones. This is to be used when trying to equilibrate a structure to its lowest energy configuration (*ionmov*=2), or in case of molecular dynamics (*ionmov*=1)

A value ten times smaller than *tolmxf* is suggested (for example 5.0×10^{-6} Hartree/Bohr).

This stopping criterion is not allowed for RF calculations.

7.1.28 tolvrs

Mnemonics: TOLerance on the potential V(r) ReSidual
Characteristic:
Variable type: real parameter
Default is 0.0 (stopping condition ignored)

Sets a tolerance for potential residual that, when reached, will cause one SCF cycle to stop (and ions to be moved).

If set to zero, this stopping condition is ignored.

Effective only when SCF cycles are done (*iscf* > 0). In this case, since *toldfe*, *toldff*, *tolvrs* and *tolwfr* are aimed at the same goal (causing the SCF cycle to stop), one and only one of these must be specified.

To get accurate stresses may be quite demanding.

7.1.29 tolwfr

Mnemonics: TOLerance on WaveFunction squared Residual
Characteristic:
Variable type: real parameter
Default is 0.0d0 (stopping criterion ignored)

Gives a convergence tolerance for the largest squared "residual" (defined below) for any given band. The squared residual is:

$$< nk|(H - E)^2|nk >, E =< nk|H|nk >, \tag{7.2}$$

which clearly is nonnegative and goes to 0 as the iterations converge to an eigenstate. With the squared residual expressed in Hartrees2 (Hartrees squared), the largest squared residual (called residm) encountered over all bands and k–points must be less than *tolwfr* for iterations to halt due to successful convergence.

Note that if *iscf* > 0, this criterion should be replaced by those based on *toldfe* (preferred for *ionmov*==0), *toldff* (preferred for *ionmov* $\neq 0$) or *tolvrs* (preferred for theoretical reasons!).

When *tolwfr* is 0.0, this criterion is ignored, and a finite value of *toldfe*, *toldff* or *tolvrs* must be specified. This also imposes a restriction on taking an ion step; ion steps are not permitted unless the largest squared residual is less than *tolwfr*, ensuring accurate forces.

To get accurate stresses may be quite demanding.

Note that the preparatory GS calculations before a RF calculations must be highly converged.

Typical values for these preparatory runs are *tolwfr* between 1.0×10^{-16} and 1.0×10^{-22}.

Note that *tolwfr* is often used in the test cases, but this is *tolwfr* purely for historical reasons: except when *iscf* < 0, other critera should be used.

7.1.30 typat

Mnemonics: TYPE of atoms
Characteristic:
Variable type: integer array typat(*natom*) (or: typat(*natrd*), if the geometry builder is used)
Default is 1 (for *natom*=1)

Array giving an integer label to every atom in the unit cell to denote its type.

The different types of atoms are constructed from the pseudopotential files. There are at most *ntypat* types of atoms. As an example, for BaTiO3, where the pseudopotential for Ba is number 1, the one of Ti is number 2, and the one of O is number 3, the actual value of the *typat* array might be:

```
typat 1 2 3 3 3
```

The array *typat* has to agree with the actual locations of atoms given in *xred*, *xcart* or *xangst*, and the input of pseudopotentials has to be ordered to agree with the atoms identified in *typat*.

The nuclear charge of the elements, given by the array *znucl*, also must agree with the type of atoms designated in "*typat*". The array *typat* is not constrained to be increasing. An internal representation of the list of atoms, deep in the code (array atindx), groups the atoms of same type together. This should be transparent to the user, while keeping efficiency.

7.1.31 udtset

Mnemonics: Upper limit on DaTa SETs
Characteristic:
Variable type: integer array udtset(2)
No Default (since it is not used when it is not defined).

Used to define the set of indices in the multi–data set mode, when a double loop is needed (see later).

The values of *udtset* must be between 1 and 9, and their product must be equal to *ndtset*.

If *udtset* is used, the input variable *jdtset* cannot be used.

7.1.32 wtk

Mnemonics: WeighTs for K points
Characteristic:
Variable type: real array wtk(*nkpt*)
Default value is *nkpt**1.0d0.

Gives the k–point weights.

The k–point weights will have their sum normalized to 1 (unless *occopt*=2; see description of *occopt*) within the program and therefore may be input with any arbitrary normalization. This feature helps avoid the need for many digits in representing fractional weights such as 1/3.

wtk is ignored if *iscf* is not positive, except if *iscf* $= -3$.

7.1.33 xangst

Mnemonics: vectors (X) of atom positions in cartesian coordinates — length in ANGSTrom-
Characteristic: NOT INTERNAL
Variable type: real array xangst(3,*natom*) (or xangst(3,*natrd*) if the geometry builder is used)

Gives the cartesian coordinates of atoms within unit cell, in angstrom. This information is redundant with that supplied by array *xred* or *xcart*.

If *xred* and *xangst* are ABSENT from the input file and *xcart* is provided, then the values of *xred* will be computed from the provided *xcart* (i.e. the user may use *xangst* instead of *xred* or *xcart* to provide starting coordinates).

One and only one of *xred*, *xcart* and *xangst* must be provided.

The conversion factor between Bohr and Åis 1 Bohr=0.5291772083 Å(See Physics Today August 1989 p.8).

Atomic positions evolve if *ionmov* $\neq 0$. In constrast with *xred* and *xcart*, *xangst* is not internal.

7.1.34 xcart

Mnemonics: vectors (X) of atom positions in CARTesian coordinates
Characteristic: EVOLVING, LENGTH
Variable type: real array *xcart*(3,*natom*) (or *xcart*(3,*natrd*) if the geometry builder is used)

Gives the cartesian coordinates of atoms within unit cell. This information is redundant with that supplied by array *xred* or *xangst*. By default, *xcart* is given in bohr atomic units (1 bohr=0.5291772083 Å), although Angstrom can be specified, if preferred, since *xcart* has the 'LENGTH' characteristics.

If *xred* and *xangst* are ABSENT from the input file and *xcart* is provided, then the values of *xred* will be computed from the provided *xcart* (i.e. the user may use *xcart* instead of *xred* or *xangst* to provide starting coordinates).

Atomic positions evolve if *ionmov* $\neq 0$.

7.1.35 xred

Mnemonics: vectors (X) of atom positions in REDuced coordinates
Characteristic: EVOLVING
Variable type: real array xred(3,*natom*) (or xred(3,*natrd*) if the geometry builder is used)
Default to all 0.0d0

Gives the atomic locations within unit cell in coordinates relative to real space primitive translations (NOT in cartesian coordinates). Thus these are fractional numbers typically between 0 and 1 and are dimensionless. The cartesian coordinates of atoms are given by: $t_{cartesian} = t1 * r1 * a1 + t2 * r2 * a2 + t3 * r3 * a3$, where (t1,t2,t3) are the "reduced coordinates" given in columns of "*xred*", (r1,r2,r3) are the columns of dimensionless array "*rprim*", and (a1,a2,a3) are the elements of the array "*acell*" giving length scales in bohr.

If you prefer to work only with cartesian coordinates, you may work entirely with "*xcart*" or "*xangst*" and ignore *xred*, in which case *xred* must be absent from the input file. One and only one of *xred*, *xcart* and Atomic positions evolve if *ionmov* $\neq 0$.

7.1.36 znucl

Mnemonics: charge -Z- of the NUCLeus
Characteristic: NO MULTI
Variable type: real array znucl(*npsp*)

Gives nuclear charge for each type of pseudopotential, in order.

If *znucl* does not agree with nuclear charge, as given in pseudopotential files, the program writes an error message and stops.

N.B.: In the pseudopotential files, *znucl* is called "zatom".

7.2 Developpement variables, VARDEV

7.2.1 accesswff

Mnemonics: ACCESS to WaveFunction Files
Characteristic: DEVELOP
Variable type: integer parameter
Default is 0.

Governs the method of access to the internal wavefunction files. Relevant only for the wavefunctions files for which the corresponding "*mkmem*"-type variable is zero, that is, for the wavefunctions that are not kept in core memory.

- 0 ⇒ Use standard Fortran IO routines
- 1 ⇒ Use MPI/IO routines (this option is only available in parallel)
- 2 ⇒ Use NetCDF routines (this option is not yet available)

The MPI/IO routines might be much more efficient than usual Fortran IO routines in the case of a large number of processors, with a pool of disks attached globally to the processors, but not one disk attached to each processor. For a cluster of workstations, where each processor has his own temporaries, the use of *accesswff*=0 might be perfectly allright.

7.2.2 ceksph

Mnemonics: CEnter K SPHere
Characteristic: DEVELOP
Variable type: integer parameter
Default is 0.

Control the set of plane waves in a sphere, generated for each k–point.

- 0 ⇒ do not center the sphere on Gamma
- 1 ⇒ do center the sphere on Gamma (this option is allowed only in the program newsp, not in abinis or abinip)

The value 0 is desirable for all usual band structure calculation, since this choice allows the symmetry to be preserved at each k–points, so that degeneracies are correct. The value 1 is used to generate input wavefunctions to the GW code of Rex Gody and coworkers. This option is only allowed in newsp.

7.2.3 dedlnn

Mnemonics:
Characteristic: ENERGY
Variable type: real parameter
Default dedlnn is 0, i.e. no correction.

Gives a value for derivative d(Etotal)/d(log(Npw)) for given value of *ecut*. Here "log" refers to a natural, base "e" logarithm. Since Etotal is an energy, *dedlnn* is also an energy. Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV).

dedlnn is used to compute the Pulay correction to the stress tensor using: correction=(1/ucvol)**dedlnn*.

See the discussion on the stress tensor given below.

This value must be computed independently by making several runs at fixed geometry and variable *ecut*, generally within +/− 3% of the desired *ecut*, and using the Etotal(npw) data to compute the derivative.

NOTE: ABINIT computes the stress tensor whenever a self–consistent energy run is performed, but the values along the diagonal of the stress tensor can have large systematic errors unless a user–provided value of *dedlnn* is input so that the appropriate Pulay correction to the diagonal stress tensor is computed.

An alternative (and more elegant) way to correct these systematic errors is provided through the use of the *ecutsm* input variable.

7.2.4 densty

Mnemonics: initial DENSity for each TYpe of atom
Characteristic: DEVELOP
Variable type: real array densty(*ntypat*)
Default is 0.0d0.

Gives a rough description of the initial GS density, for each type of atom. This value is only used to create the first exchange and correlation potential, and is not used anymore afterwards.

For the time being, it corresponds to an average radius (a.u.) of the density, and is used to generate a gaussian density. If set to 0.0d0, an optimized value is used.

No meaning for RF calculations.

7.2.5 effmass

Mnemonics: EFFective MASS
Characteristic: DEVELOP
Variable type: real number
Default value is one.

This parameter allows to change the electron mass, with respect to its experimental value.

7.2.6 eshift

Mnemonics: Energy SHIFT
Characteristic: DEVELOP, ENERGY
Variable type: real number
Default value is zero.

Used only if *wfoptalg*=3. *eshift* gives the shift of the energy used in the shifted Hamiltonian squared. The algorithm will determine eigenvalues and eigenvectors centered on *eshift*.

Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV)

7.2.7 exchn2n3

Mnemonics: EXCHange N2 and N3
Characteristic: DEVELOP

Variable type: integer parameter
Default is 0.

If *exchn2n3* is 1, the internal representation of the FFT arrays in reciprocal space will be array(n1,n3,n2), where the second and third dimensions have been switched. This is to allow to be coherent with the *exchn2n3*=4xx FFT treatment.

7.2.8 fftalg

Mnemonics: Fast Fourier Transform ALGorithm
Characteristic: DEVELOP
Variable type: integer parameter
Default is 112, except for VPP Fujitsu, for which the Default is 111, and for NEC, for which the default is 200.

Allows to choose the algorithm for Fast Fourier Transforms. These have to be used when applied to wavefunctions (routine fourwf.f), as well as when applied to densities and potentials (routine fourdp.f). Presently, it is the concatenation of three digits, labelled (A), (B) and (C).
The first digit (A) is to be chosen among 1, 2, 3 and 4:

- 1 ⇒ use FFT routines written by S. Goedecker.
- 2 ⇒ use machine–dependent FFT algorithm, taken from the vendor library, if it exists and if it has been implemented. The bare *fftalg*=200 has little chance to be faster than *fftalg*=112, but it might be tried. Implementing library subroutines with *fftalg* ≠ 200 has not yet been done. Currently implemented library subroutines (*fftalg*=200) are:
  - on HP, z3dfft from Veclib;
  - on DEC Alpha, zfft_3d from DXML;
  - on NEC, ZFC3FB from ASL lib;
  - on SGI, zfft3d from complib.sgimath
- 3 ⇒ use serial or multi–threaded FFTW fortran routines (http://www.fftw.org). Currently implemented with *fftalg*=300.
- 4 ⇒ use FFT routines written by S. Goedecker, 2002 version, that will be suited for MPI and OpenMP parallelism.

The second digit (B) is related to fourdp.f:

- 0 ⇒ only use Complex–to–complex FFT
- 1 ⇒ real–to–complex is also allowed (only coded for A==1)

The third digit (C) is related to fourwf.f:

- 0 ⇒ no use of zero padding
- 1 ⇒ use of zero padding (only coded for A==1 and A==4)
- 2 ⇒ use of zero padding, and also combines actual FFT operations (using 2 routines from S. Goedecker) with important pre– and post–processing operations, in order to maximize cache data reuse. This is very efficient for cache architectures. (coded for A==1 and A==4, but A==4 is not yet sufficiently tested)

Internal representation as *ngfft*(7).

7.2.9 fftcache

Mnemonics: Fast Fourier Transform CACHE size
Characteristic: DEVELOP
Variable type: integer parameter
Default is 16. Not yet machine–dependent.

Gives the cache size of the current machine, in Kbytes.
Internal representation as *ngfft*(8).

7.2.10 freqsusin

Mnemonics: FREQuencies for the SUSceptibility matrix: the INcrement
Characteristic: DEVELOP
Variable type: real parameter, positive or zero
Default is 0.0

Define, with freqsuslo, the series of imaginary frequencies at which the susceptibility matrix should be computed.
This is still under development.

7.2.11 freqsuslo

Mnemonics: FREQuencies for the SUSceptibility matrix: the LOwest frequency
Characteristic: DEVELOP
Variable type: real parameter, positive or zero
Default is 0.0

Define, with *freqsusin*, the series of imaginary frequencies at which the susceptibility matrix should be computed.
This is still under development.

7.2.12 idyson

Mnemonics: Integer giving the choice of method for the DYSON equation
Characteristic: DEVELOP
Variable type: integer parameter
Default value is 1.

Choice for the method used to solve the Dyson equation in the calculation of the interacting susceptibility matrix or/and in the calculation of the ACFD exchange–correlation energy:

- *idyson*=1: Solve the Dyson equation by direct matrix inversion
- *idyson*=2: Solve the Dyson equation as a first–order differential equation with respect to the coupling constant lambda — only implemented for the RPA at the present stage (see header of dyson_de.f for details)
- *idyson*=3: Calculate only the diagonal of the interacting susceptibility matrix by self–consistently computing the linear density change in response to a set of perturbations. Only implemented for the RPA at the present stage, and entirely experimental (see dyson_sc.f for details).

7.2.13 ikhxc

Mnemonics: Integer option for KHXC = Hartree XC kernel
Characteristic:
Variable type: integer parameter
Default value is 1.

Define the HXC kernel, in the cases for which it can be dissociated with the choice of the HXC functional given by *ixc*, namely the TD–DFT computation of excited states (*iscf* $= -1$), and the computation of the susceptibility matrix (for ACFD purposes). Options 2 to 6 are for the ACFD only.

- 0 ⇒ RPA for the TDDFT but no kernel for the ACFD (testing purposes).
- 1 ⇒ RPA for the TDDFT and ACFD.
- 2 ⇒ ALDA (PW92) for the ACFD
- 3 ⇒ PGG for the ACFD [M. Petersilka, U.J. Gossmann and E.K.U. Gross, PRL 76,1212 (1996)]
- 4 ⇒ BPG for the ACFD. This amounts to half the PGG kernel plus half the ALDA kernel for spin–compensated systems [K. Burke, M. Petersilka and E.K.U. Gross, in "Recent Advances in Density Functional Methods", Vol. III, edited by P. Fantucci and A. Bencini (World Scientific, Singapore, 2002)]
- 5 ⇒ Linear energy optimized kernel [J. Dobson and J. Wang, PRB 62, 10038 (2000)]
- 6 ⇒ Non–linear energy optimized kernel [J. Dobson and J. Wang, PRB 62, 10038 (2000)]

For ACFD–ALDA, BPG and energy optimized kernels are highly experimental and not tested yet!!! For ACFD calculations, a cut–off density has been defined for the ALDA, BPG and energy optimized kernels: let rhomin = *userre**rhomax (where rhomax is the maximum density in space); then the actual density used to calculate the local part of these kernels at point r is max(rho(r),rhomin.

7.2.14 intexact

Mnemonics: INTegration using an EXACT scheme
Characteristic: DEVELOP
Variable type: integer parameter
Default value is 0.

Relates to the ACFD xc functionals only. If *intexact* > 0, the integration over the coupling constant will be performed analytically in the RPA and in the two–electron PGG approximation for the ACFD exchange–correlation energy. Otherwise, the integration over the coupling constant will be performed numerically (also see *ndyson* and *idyson*. Note that the program will stop in *intexact* > 0 and *ikhxc* $\neq 1$ (RPA) or *ikhxc* $\neq 3$ (PGG, with two electrons)

7.2.15 intxc

Mnemonics: INTerpolation for eXchange–Correlation
Characteristic: DEVELOP
Variable type: integer parameter
Default value is 0.

- 0 ⇒ do "usual" xc quadrature on fft grid

- 1 ⇒ do higher accuracy xc quadrature using fft grid and additional points at the centers of each cube (doubles number of grid points)–the high accuracy version is only valid for boxcut≥2. If boxcut ¡ 2, the code stops.

For RF calculations only *intxc*=0 is allowed yet. Moreover, the GS preparation runs (giving the density file and zero–order wavefunctions) must be done with *intxc*=0

Prior to ABINITv2.3, the choice *intxc*=1 was favoured (it was the default), but the continuation of the development of the code lead to prefer the default *intxc*=0. Indeed, the benefit of *intxc*=1 is rather small, while making it available for all cases is a non–negligible development effort. Other targets are prioritary ... You will notice that many automatice tests use *intxc*=1. Please, do not follow this historical choice for your production runs.

7.2.16 iprcch

Mnemonics: Integer for PReConditioning of CHarge response
Characteristic: DEVELOP
Variable type: integer parameter
Default for iprcch is 2, unless *ionmov*=4 and *iscf*=5, in which case iprcch is automatically put to 3.

Used when *iscf* > 0, to define the SCF preconditioning scheme. Potential–based preconditioning schemes for the SCF loop are still under development. The present parameter (charge part: mixed electronic–atomic) describe the way a change of density is derived from a change of atomic position. Supported values:

- 0 ⇒ fixed charge
- 1 ⇒ rigid ion hypothesis (atomic charge moves with atom) used to correct the forces
- 2 ⇒ rigid ion hypothesis (atomic charge moves with atom) used to correct forces and density
- 3 ⇒ a different implementation of the rigid ion hypothesis (atomic charge moves with atom) used to correct forces and density

For the time being, the choice 3 must be used with *ionmov*=4 and *iscf*=5. Otherwise, use the choice 2.

No meaning for RF calculations.

7.2.17 iprcfc

Mnemonics: Integer for PReConditioner of Force Constants
Characteristic: DEVELOP
Variable type: integer parameter
Default for iprcfc is 0.

Used when *iscf* > 0, to define the SCF preconditioning scheme. Potential–based preconditioning schemes for the SCF loop are still under development.

The present parameter (force constant part) describe the way the a change of force is derived from a change of atomic position.

Supported values:

- 0 ⇒ hessian is the identity matrix
- 1 ⇒ hessian is 0.5 times the identity matrix
- 2 ⇒ hessian is 0.25 times the identity matrix
- −1 ⇒ hessian is twice the identity matrix
- ... (simply corresponding power of 2 times the identity matrix)

No meaning for RF calculations.

7.2.18 isecur

Mnemonics: Integer for level of SECURity choice
Characteristic: DEVELOP
Variable type: integer
Default is 0.

In the presently used algorithms, there is a compromise between speed and robustness, that can be tuned by using *isecur*.

If *isecur*=0, an extrapolation of out–of–line data is allowed, and might save one non–SCF calculation every two line minimisation when some stability conditions are fulfilled (since there are 2 non–SCF calculations per line minimisation, 1 out of 4 is saved)

Using *isecur*=1 or higher integers will raise gradually the threshold to make extrapolation.

Using *isecur* = −2 will allow to save 2 non–SCF calculations every three line minimisation, but this can make the algorithm unstable. Lower values of *isecur* allows for more (tentative) savings. In any case, there must be one non–SCF computation per line minimisation.

No meaning for RF calculations yet.

7.2.19 istatr

Mnemonics: Integer for STATus file repetition Rate

7.2.20 istatshft

Mnemonics: Integer for STATus file SHiFT

Characteristic: DEVELOP, NO MULTI
Variable type: integer parameter
Default *istatr* is 49, and 149 for Cray T3E (slow I/Os). Values lower than 10 may not work on some machines. Default *istatshft* is 1.

Govern the rate of output of the status file. This status file is written when the number of the call to the status subroutine is equal to '*istatshft*' modulo '*istatr*', so that it is written once every '*istatr*' call. There is also a writing for each of the 5 first calls, and the 10th call.

7.2.21 istwfk

Mnemonics: Integer for choice of STorage of WaveFunction at each k point
Characteristic:
Variable type: integer array istwfk(*nkpt*)
Default is 0 for all k–points for GS calculations. For RF calculations, the Default is not used: istwfk is forced to be 1 deep inside the code, for all k–points. For spin–orbit calculations (*nspinor*=2), istwfk is also forced to be 1, for all k–points.

Control the way the wavefunction for each k–point is stored inside ABINIT, in reciprocal space.

For the GS calculations, in the "cg" array containing the wavefunction coefficients, there is for each k–point and each band, a segment cg(1:2,1:npw). The 'full' number of plane wave is determined by *ecut*. However, if the k–point coordinates are build only from zeroes and halves (see list below), the use of time–reversal symmetry (that connects coefficients) has been implemented, in order to use real–to–complex FFTs (see *fftalg*), and to treat explicitly only half of the number of plane waves (this being used as 'npw').

For the RF calculations, there is not only the "cg" array, but also the "cgq" and "cg1" arrays. For the time–reversal symmetry to decrease the number of plane waves of these arrays, the q vector MUST be (0 0 0). Then, for each k–point, the same rule as for the RF can be applied.

WARNING (991018): for the time being, the time–reversal symmetry cannot be used in the RF calculations.

- 1 ⇒ do NOT take advantage of the time–reversal symmetry
- 2 ⇒ use time–reversal symmetry for k=(0 0 0)
- 3 ⇒ use time–reversal symmetry for k=(1/2 0 0)
- 4 ⇒ use time–reversal symmetry for k=(0 0 1/2)
- 5 ⇒ use time–reversal symmetry for k=(1/2 0 1/2)
- 6 ⇒ use time–reversal symmetry for k=(0 1/2 0)
- 7 ⇒ use time–reversal symmetry for k=(1/2 1/2 0)
- 8 ⇒ use time–reversal symmetry for k=(0 1/2 1/2)
- 9 ⇒ use time–reversal symmetry for k=(1/2 1/2 1/2)
- 0 ⇒ (preprocessed) for each k–point, choose automatically the appropriate time–reversal option when it is allowed, and chose *istwfk*=1 for all the other k–points.

Note that the input variable "*mkmem*" also controls the wavefunction storage, but at the level of core memory versus disk space.

7.2.22 ldgapp

Mnemonics: Lein–Dobson–Gross approximation
Characteristic: DEVELOP
Variable type: integer parameter
Default is 0.

Concern only the ACFD computation of the correlation energy (*optdriver*=3).

If *ldgapp* > 0, the Lein, Dobson and Gross first–order approximation to the correlation energy is also computed during the ACFD run. [See Lein, Dobson and Gross, J. Comput. Chem. 20,12 (1999)].

This is only implemented for the RPA, for the PGG kernel and for the linear energy optimized kernel at the present time.

7.2.23 mqgrid

Mnemonics: Maximum number of Q–space GRID points for pseudopotentials
Characteristic: DEVELOP
Variable type: integer parameter
Default is 1201.

Govern the size of the one–dimensional information related to pseudopotentials, in reciprocal space: potentials, or projector functions.

7.2.24 nbandsus

Mnemonics: Number of BANDs to compute the SUSceptibility
Characteristic:
Variable type: integer parameter
Default value is *nband*.

Number of bands to be used in the calculation of the susceptibility matrix (ACFD only).

7.2.25 nbdblock

Mnemonics: Number of BanDs in a BLOCK
Characteristic: DEVELOP
Variable type: integer parameter
Default is 1

In case of non–standard, blocked algorithms for the optimization of the wavefunctions (that is, if *wfoptalg* $\neq 0$), *nbdblock* defines the number of bands (or states) in a block.

7.2.26 ndyson

Mnemonics: Number of points to be added for the solution of the DYSON equation
Characteristic:
Variable type: integer parameter
Default value is -1.

Number of points to be added to lambda=0 and lambda=1 (that are always calculated for the integration ober the coupling constant lambda in the ACFD calculation of the exchange–correlation energy.

- *ndyson* $= -1$: let the code decide how many points to use (presently, 3 points for *idyson*=1 or 3, and 9 points for *idyson*=2)
- *ndyson* $= 0$: only compute the non–interacting and fully–interacting susceptibility matrices.
- *ndyson* > 0: use *ndyson* more points in]0,1[

7.2.27 nfreqsus

Mnemonics: Number of FREQuencies for the SUSceptibility matrix
Characteristic: DEVELOP
Variable type: integer parameter
Default is 0

If 0, no computation of frequency–dependent susceptibility matrix. If 1 or larger, will read freqsuslo and *freqsusin* to define the frequencies (1 is currently the only value allowed)

7.2.28 nloalg

Mnemonics: Non Local ALGorithm
Characteristic: DEVELOP
Variable type: integer variable
Default is 4, except for the NEC where it is 2.

Allows to choose the algorithm for non–local operator application. On super–scalar architectures, the Default *nloalg*=4 is the best, but you can save memory by using *nloalg* $= -4$. More detailed explanations:

- *nloalg*=2: Should be efficient on vector machines. It is indeed the fastest algorithm for the NEC, but actual tests on Fujitsu machine did not gave better performances than the other options.
- *nloalg*=3: same as *nloalg*==2, but the loop order is inverted.
- *nloalg*=4: same as *nloalg*==3, but maximal use of registers has been coded. This should be especially efficient on scalar and super–scalar machines. This has been confirmed by tests.

Negative values of *nloalg* correspond positive ones, where the phase precomputation has been suppressed, in order to save memory space: an array double precision :: ph3d(2,npw,*natom*) is saved (typically half the space needed for the wavefunctions at 1 k–point — this corresponds to the silicon case). However, the computation of phases inside nonlop is somehow time–consuming.

Note: internally, *nloalg* is an array *nloalg*(1:4), that also allows to initialize, in order, jump, mblkpw, and mincat (not documented). However, only the first component *nloalg*(1) is read as an input variable.

7.2.29 nnsclo

Mnemonics: Number of Non–Self Consistent LOops
Characteristic: DEVELOP
Variable type: integer parameter
Default is 0.

Gives the maximum number of non–self–consistent loops of *nline* line minimisations, in the SCF case (when *iscf* > 0). In the case *iscf* ≤ 0, the number of non–self–consistent loops is determined by *nstep*.

The Default value of 0 correspond to make the two first fixed potential determinations of wavefunctions have 2 non–self consistent loops, and the next ones to have only 1 non–self consistent loop.

7.2.30 optforces

Mnemonics: OPTions for the calculation of FORCES
Characteristic: DEVELOP
Variable type: integer parameter
Default is 1.

Allows to choose options for the calculation of forces.

- *optforces*=0: the forces are set to zero, and many steps of the computation of forces are skipped
- *optforces*=1: calculation of forces at each SCF iteration, allowing to use forces as criterion to stop the SCF cycles
- *optforces*=2: calculation of forces at the end of the SCF iterations (like the stresses) — NOT YET IMPLEMENTED

7.2.31 ortalg

Mnemonics: ORThogonalisation ALGorithm
Characteristic: DEVELOP
Variable type: integer parameter
Default is 2.

Allows to choose the algorithm for orthogonalisation.

Positive or zero values make two projections per line minimisation, one before the preconditioning, one after. This is the clean application of the band–by–band CG gradient for finding eigenfunctions.

Negative values make only one projection per line mininisation.

The orthogonalisation step is twice faster, but the convergence is less good. This actually calls to a better understanding of this effect.

ortalg=0, 1 or −1 is the conventional coding, actually identical to the one in versions prior to 1.7

ortalg=2 or −2 try to make better use of existing registers on the particular machine one is running.

More demanding use of registers is provided by *ortalg* = 3 or −3, and so on.

The maximal value is presently 4 and −4.

Tests have shown that *ortalg* = 2 or −2 is suitable for use on the available platforms.

7.2.32 qprtrb

Mnemonics: Q–wavevector of the PERTurbation
Characteristic: DEVELOP
Variable type: integer array of three values
Default wavevector is 0 0 0.

Gives the wavevector, in units of reciprocal lattice primitive translations, of a perturbing potential of strength *vprtrb*. See *vprtrb* for more explanation.

7.2.33 useria, userib, useric, userid, userie

Mnemonics: USER Integer variables A, B, C, D and E
Characteristic:
Variable type: integers
Default values are 0.

These are user–definable integers which the user may input and then utilize in subroutines of his/her own design. They are not used in the official versions of the ABINIT code, and should ease independent developments (hopefully integrated in the official version afterwards).

Internally, they are available in the dtset structured datatype, e.g. dtset%*useria*.

7.2.34 userra, userrb, userrc, userrd, userre

Mnemonics: USER Real variables A, B, C, D, and E
Characteristic:
Variable type: real numbers

These are user–definable with the same purpose as useri above.

Default value is 0.0.

7.2.35 useylm

Mnemonics: USE YLM (the spherical harmonics)
Characteristic: DEVELOP
Variable type: integer parameter
Default is 0.

(Not working yet: purely for developpers)

7.2.36 vprtrb

Mnemonics: potential -V- for the PeRTuRBation
Characteristic: DEVELOP, ENERGY
Variable type: real array of 2 elements

Default value is 0.d0 0.d0.

Gives the real and imaginary parts of a scalar potential perturbation. Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics.
This is made available for testing responses to such perturbations. The form of the perturbation, which is added to the local potential, is:

- (*vprtrb*(1)+I**vprtrb*(2))/2 at G=*qprtrb* and
- (*vprtrb*(1)-I**vprtrb*(2))/2 at G=-*qprtrb* (see *qprtrb* also).

7.2.37 wfoptalg

Mnemonics: WaveFunction OPTimisation ALGorithm
Characteristic: DEVELOP
Variable type: integer parameter
Default is 0.

Allows to choose the algorithm for the optimisation of the wavefunctions. The different possibilities are:

- *wfoptalg*=0: standard state–by–state conjugate gradient algorithm, with no possibility to parallelize over the states;
- *wfoptalg*=1: blocked conjugate gradient algorithm, with possibility to parallelize over the states (or bands), but at the expense of a few more operations when a block of states has been optimized separately, to obtain a coherent set of wavefunctions. The number of states in a block is defined in *nbdblock*
- *wfoptalg*=2: minimisation of the residual with respect to different shifts, in order to cover the whole set of occupied bands, with possibility to parallelize over blocks of states (or bands). The number of states in a block is defined in *nbdblock*. THIS IS STILL IN DEVELOPMENT.
- *wfoptalg*=3: minimisation of the residual with respect to a shift. Available only in the non–self–consistent case *iscf* = −2, in order to find eigenvalues and wavefunctions close to a prescribed value.

7.3 Files handling variables, VARFIL

7.3.1 cmlfile

Mnemonics: Chemical Markup Language FILE
Characteristic: NO INTERNAL
Variable type: character string
Default is no file.

Used to import some of the data from one or more Chemical Markup Language 2 (CML2) file(s) (one per dataset). Unlike most of the other input variables, it refers to a character string, e.g.:

```
cmlfile ../t67.in_CML.xml
```

The file is preprocessed, and the relevant information is translated in order to be used as an alternative to the usual input variables. Note that the input variables directly defined in the usual input file have precedence over the CML data: the latter are used only when there is no occurence of the corresponding keyword in the input file ...

The ABINIT CML parser is still quite primitive. The mechanism followed to parse the CML file is described afterwards.

The ABINIT CML parser will localize in the CML file the first occurence of a 'molecule' markup section. It will ignore all other occurences of 'molecule'. Inside this 'molecule' section, it will localize the first occurences of the 'crystal', 'symmetry' and 'atomArray' sections. All other occurences, and all other sections, are ignored.

The following ABINIT input variables will be extracted from these sections of the CML file (if the data is available):

- *acell* from the first 'scalar title="a"', 'scalar title="b"', and 'scalar="c"' sections (all three must be present if one is present) in the 'crystal' section, expecting the data in Angstrom;
- *angdeg* from the first 'scalar title="alpha"', 'scalar title="beta"', and 'scalar title="gamma"' sections (all three must be present if one is present) in the 'crystal' section, expecting the data in degrees;
- *nsym*, *symrel* and *tnons* from the content of 'matrix' sections in the 'symmetry' section;
- *natom* from the number of items in the first 'atom' sections in the 'atomArray' section;
- *typat* from the attribute 'elementType' in the 'atom' sections in the 'atomArray' section, with identification of the pseudopotentials that have the correct nuclear charge, according to the atomic symbol (the first pseudopotential with the correct nuclear charge, from the pseudopotential list, will be used);
- *xred* from the attributes 'xFract', 'yFract', and 'zFract' (all three must be present if one is present) in the 'atom' sections in the 'atomArray' section.

These limited parsing capabilities are enough for ABINIT to read the CML files it has created thanks to the use of the *prtcml* input variable.

7.3.2 getden

Mnemonics: GET the DENsity from ...
Characteristic:
Variable type: integer parameter
Default is 0.

Eventually used when $ndtset > 0$ (multi–dataset mode) and, in the case of a ground–state calculation, if $iscf < 0$ (non–SCF calculation), to indicate that the starting density is to be taken from the output of a previous dataset. It is used to chain the calculations, since it describes from which dataset the OUTPUT density are to be taken, as INPUT density of the present dataset.

If *getden*==0, no such use of previously computed output density file is done.

If *getden* is positive, its value gives the index of the dataset from which the output density is to be used as input.

If *getden* is -1, the output density of the previous dataset must be taken, which is a frequently occuring case.

If *getden* is a negative number, it indicates the number of datasets to go backward to find the needed file. In this case, if one refers to a non existant data set (prior to the first), the density is not initialised from a disk file, so that it is as if $getden = 0$ for that initialisation. Thanks to this rule, the use of $getden\ -1$ is rather straightforward: except for the first density, that is not initialized by reading a disk file, the output density of one dataset is input of the next one.

Be careful: the output density file of a run with non–zero *ionmov* does not have the proper name (it has a "TIM" indication) for use as an input of an $iscf < 0$ calculation.

One should use the output density of a *ionmov*==0 run.

7.3.3 getkss

Mnemonics: GET Kohn–Sham Structure from ...
Characteristic: GW
Variable type: integer parameter
Default is 0.

Used when *ndtset* > 0 (multi–dataset mode) and *optdriver*=3 or 4 (a GW calculation), to indicate that the KSS wavefunction file is to be taken from the output of a previous dataset. It is used to chain the calculations, since it describes from which dataset the OUTPUT wavefunctions are to be taken, as INPUT of the present dataset.

If *getkss*==0, no such use of previously computed output KSS file is done.

If *getkss* is positive, its value gives the index of the dataset from which the output KSS file is to be used as input.

If *getkss* is −1, the output KSS file of the previous dataset must be taken, which is a frequently occuring case.

If *getkss* is a negative number, it indicates the number of datasets to go backward to find the needed file. In this case, if one refers to a non existent data set (prior to the first), the KSS file is not initialised from a disk file, so that it is as if *getkss*=0 for that initialisation.

7.3.4 getocc

Mnemonics: GET OCC parameters from ...
Characteristic:
Variable type: integer parameter, an instance of a 'get' variable
Default is 0.

This variable is typically used to chain the calculations, in the multi–dataset mode (*ndtset* > 0), since it describes from which dataset the array *occ* is to be taken, as input of the present dataset. The occupation numbers are EVOLVING variables, for which such a chain of calculations is useful.

If ==0, no use of previously computed values must occur.

If it is positive, its value gives the index of the dataset from which the data are to be used as input data. It must be the index of a dataset already computed in the SAME run.

If equal to −1, the output data of the previous dataset must be taken, which is a frequently occuring case. However, if the first dataset is treated, −1 is equivalent to 0, since no dataset has yet been computed in the same run.

If another negative number, it indicates the number of datasets to go backward to find the needed data (once again, going back beyond the first dataset is equivalent to using a null get variable).

NOTE that a non–zero *getocc* MUST be used with *occopt*==2, so that the number of bands has to be initialized for each k–point. Of course, these numbers of bands must be identical with the numbers of bands of the dataset from which *occ* will be copied. The same is true for the number of k–points.

7.3.5 getscr

Mnemonics: GET SCReening (the inverse dielectric matrix) from ...
Characteristic: GW
Variable type: integer parameter
Default is 0.

Used when *ndtset* > 0 (multi–dataset mode) and *optdriver*=4 (sigma step of a GW calculation), to indicate that the dielectric matrix (EPS file) is to be taken from the output of a previous dataset.

It is used to chain the calculations, since it describes from which dataset the OUTPUT dielectric matrix are to be taken, as INPUT of the present dataset.

If *getscr*==0, no such use of previously computed output EPS file is done.

If *getscr* is positive, its value gives the index of the dataset from which the output EPS file is to be used as input.

If *getscr* is −1, the output EPS file of the previous dataset must be taken, which is a frequently occuring case.

If *getscr* is a negative number, it indicates the number of datasets to go backward to find the needed file. In this case, if one refers to a non existent data set (prior to the first), the EPS file is not initialised from a disk file, so that it is as if *getscr*=0 for that initialisation.

7.3.6 getwfk

Mnemonics: GET the wavefunctions from _WFK file

7.3.7 getwfq

Mnemonics: GET the wavefunctions from _WFQ file

7.3.8 get1wf

Mnemonics: GET the first–order wavefunctions from _1WF file

7.3.9 getddk

Mnemonics: GET the ddk wavefunctions from _1WF file
Characteristic:
Variable type: integer parameter
Default is 0.

Eventually used when *ndtset* > 0 (in the multi–dataset mode), to indicate starting wavefunctions, as an alternative to *irdwfk*, *irdwfq*, *ird1wf*, or *irdddk*. One should first read the explanations given for these latter variables.

The *getwfk*, *getwfq*, *get1wf* and *getddk* variables are typically used to chain the calculations in the multi–dataset mode, since they describe from which dataset the OUTPUT wavefunctions are to be taken, as INPUT wavefunctions of the present dataset.

We now focus on the *getwfk* input variable (the only one used in ground–state calculations), but the rules for *getwfq* and *get1wf* are similar, with _WFK replaced by _WFQ or _1WF.

If *getwfk*==0, no use of previously computed output wavefunction file appended with _DSx_WFK is done.

If *getwfk* is positive, its value gives the index of the dataset for which the output wavefunction file appended with _WFK must be used.

If *getwfk* is −1, the output wf file with _WFK of the previous dataset must be taken, which is a frequently occuring case.

If *getwfk* is a negative number, it indicates the number of datasets to go backward to find the needed wavefunction file. In this case, if one refers to a non existent data set (prior to the first), the wavefunctions are not initialised from a disk file, so that it is as if *getwfk*=0 for that initialisation. Thanks to this rule, the use of *getwfk* −1 is rather straightforward: except for the first wavefunctions, that are not initialized by reading a disk file, the output wavefunction of one dataset is input of the next one.

In the case of a ddk calculation in a multidataset run, in order to compute correctly the localisation tensor, it is mandatory to declare give *getddk* the value of the current dataset (i.e. *getddk*3 3) — this is a bit strange and should be changed in the future.

7.3.10 get1den

Mnemonics: GET the wavefunctions from _WFK file, DenSIFied?? (to be completed)

7.3.11 get1wfden

Mnemonics: GET the wavefunctions from _WFK file, DenSIFied?? (to be completed)

7.3.12 irdkss

Mnemonics: Integer that governs the ReaDing of KSS file
Characteristic: GW
Variable type: integer parameter
Default is 0.

Relevant only when *optdriver*=3 or 4. Indicate the file from which the dielectric matrix must be obtained. As alternative, one can use the input variable *getkss*.

When *optdriver*=3 or 4, at least one of *irdkss* or *getscr* must be non–zero.

A non–zero value of *irdkss* is treated in the same way as other "ird" variables, see the section 4 of `abinis_help`.

7.3.13 irdscr

Mnemonics: Integer that governs the ReaDing of EPSilon
Characteristic: GW
Variable type: integer parameter
Default is 0.

Relevant only when *optdriver*=4. Indicate the file from which the dielectric matrix must be obtained. As alternative, one can use the input variable *getscr*.

When *optdriver*=4, at least one of *irdscr* or *getscr* must be non–zero.

A non–zero value of *irdscr* is treated in the same way as other "ird" variables, see the section 4 of `abinis_help`.

7.3.14 irdwfk

Mnemonics: Integer that governs the ReaDing of _WFK files

7.3.15 irdwfq

Mnemonics: Integer that governs the ReaDing of _WFQ files

7.3.16 ird1wf

Mnemonics: Integer that governs the ReaDing of _1WF files

7.3.17 irdddk

Mnemonics: Integer that governs the ReaDing of DDK wavefunctions, in _1WF files
Characteristic:
Variable type: integer parameter
Default is 0.

Indicates eventual starting wavefunctions. As alternative, one can use the input variables *getwfk*, *getwfq*, *get1wf* or *getddk*.
Ground–state calculation:

- only *irdwfk* and *getwfk* have a meaning
- at most one of *irdwfk* or *getwfk* can be non–zero
- if *irdwfk* and *getwfk* are both zero, initialize wavefunctions with random numbers for ground state calculation.
- if *irdwfk* = 1: read ground state wavefunctions from a disk file appended with _WFK, produced in a previous ground state calculation (see the section 4 of `abinis_help`).

Response–function calculation:

- one and only one of *irdwfk* or *getwfk* MUST be non–zero
- if *irdwfk* = 1: read ground state k–wavefunctions from a disk file appended with _WFK, produced in a previous ground state calculation (see the section 4 of `abinis_help`).
- only one of *irdwfq* or *getwfq* can be non–zero, if both of them are non–zero, use as $k+q$ file the one defined by *irdwfk* and/or *getwfk*
- if *irdwfq* = 1: read ground state $k+q$–wavefunctions from a disk file appended with _WFQ, produced in a previous ground state calculation (see the section 4 of `abinis_help`).
- at most one of *ird1wf* or *get1wf* can be non–zero
- if both are zero, initialize first order wavefunctions to 0's.
- if *ird1wf* = 1: read first–order wavefunctions from a disk file appended with _1WFx, produced in a previous response function calculation (see the section 4 of `abinis_help`).
- at most one of *irdddk* or *getddk* can be non–zero
- one of them must be non–zero if an homogeneous electric field calculation is done (presently, a ddk calculation in the same dataset is not allowed)
- if *irdddk* = 1: read first–order ddk wavefunctions from a disk file appended with _1WFx, produced in a previous response function calculation (see the section 4 of `abinis_help`).

7.3.18 kssform

Mnemonics: Kohn Sham Structure file FORMat
Characteristic:
Variable type: integer parameter
Default is 1, i.e. the KSS format

Governs the choice of the format for the file that contains the Kohn–Sham electronic structure information, for use in GW calculations, see the input variables *optdriver* and *nbandkss*.

- (obsolete) *kssform*=0, the _STA file is generated together with a _VKB file containing information on the pseudopotential
- *kssform*=1, a single file `.kss` (double precision) containing complete information on the Kohn Sham Structure (eigenstates and the pseudopotentials used) will be generated through full diagonalization of the complete Hamiltonian matrix. The file has at the beginning the standard abinit header
- (obsolete) *kssform*=2, the same as 1, but most of the relevant informations are in single precision.
- *kssform*=3, a single file `.kss` (double precision) containing complete information on the Kohn Sham Structure (eigenstates and the pseudopotentials used) will be generated through the usual conjugate gradient algorithm (so, a restricted number of states) The file has at the beginning the standard abinit header

Very important: for the time being, *istwfk* must be 1 for all the k–points.

7.3.19 mffmem

Mnemonics: Maximum number of FFt grids in MEMory
Characteristic:
Variable type: integer parameter
Default is 1, i.e. an in–core solution.

Governs the choice of number of FFT arrays that will be kept permanently in core memory.

The allowed values are 0, in which case maximal use is made of disk space, saving core memory at the expense of execution time (not much, usually), or 1, in which case everything is kept in core memory.

More detailed explanations: if *mffmem*==0, some arrays of size double precision :: xx(*nfft*,*nsppol*) will be saved on disk when the wavefunctions are optimized or when the Hartree and xc potential is computed (which can require some sizeable memory space also).

The number of these arrays is 10 if *iscf*==5, 5 if *iscf*==1, and 4 if *iscf*==2 or 3. The saving of memory can be appreciable especially when *iscf*==5 and *nsppol*=2.

7.3.20 mkmem

Mnemonics: Maximum number of K - points in MEMory
Characteristic:
Variable type: integer parameter
Default is *nkpt*, i.e. in–core solution.

Sets the maximum number of k–points for which the ground state wavefunctions are kept in core memory at one time.

This value should either be 0, in which case an out–of–core solution will be used, or else *nkpt*, in which case an in–core solution will be used.

Internal representation as *mkmem*s(1)

7.3.21 prtcml

Mnemonics: PRinT CML file
Characteristic:
Variable type: integer parameter
Default is 0.

If set to 1 or a larger value, provide output of geometrical parameters using CML (the Chemical Markup Language, see papers by P. Murray–Rust and H. Rzepa, especially J. Chem. Inf. Comput. Sci. 39, 928–942 (1998) and the Web site http://www.xml-cml.org). Such file can be treated automatically by tools developed to handle XML formatted files.

Such a CML file contains:

- The crystallographic information (space group number and the needed unit cell parameters and angles)
- The list of symmetry elements
- The list of atoms in the cell (symbols and reduced coordinates)

If *ionmov*==0, the name of the CML file will be the root output name, followed by _CML.xml.

If *ionmov*==1 or 2, the CML file will be output at each time step, with the name being made of

- the root output name,
- followed by _TIMx, where x is related to the timestep (see later)
- then followed by _CML.xml

No output is provided by *prtcml* lower or equal to 0.

7.3.22 prtden

Mnemonics: PRinT the DENsity
Characteristic:
Variable type: integer parameter
Default is 0.

If set to 1 or a larger value, provide output of electron density in real space rho(r), in units of electrons/Bohr^3.

If *ionmov*==0, the name of the density file will be the root output name, followed by _DEN.

If *ionmov*==1 or 2, density files will be output at each time step, with the name being made of

- the root output name,
- followed by _TIMx, where x is related to the timestep (see later)
- then followed by _DEN

The file structure of the unformatted output file is described below, see section 6).

No output is provided by *prtden* lower or equal to 0.

7.3.23 prtdos

Mnemonics: PRinT the Density Of States
Characteristic:
Variable type: integer parameter
Default is 0.

Provide output of Density of States if set to 1, 2 or 3. Can either use a smearing technique (*prtdos*=1), or the tetrahedron method (*prtdos*=2). If *prtdos*=3, provide output of Local Density of States inside a sphere centered on an atom, as well as the angular–momentum projected DOS, in the same sphere. The resolution of the linear grid of energies for which the DOS is computed can be tuned thanks to *dosdeltae*.

If *prtdos*=1, the smeared density of states is obtained from the eigenvalues, properly weighted at each k–point using *wtk*, and smeared according to *occopt* and *tsmear*. All levels that are present in the calculation are taken into account (occupied and unoccupied). Note that *occopt* must be between 3 and 7.

In order to compute the DOS of an insulator with *prtdos*=1, compute its density thanks to a self–consistent calculation (with a non–metallic *occopt* value, 0, 1 or 2), then use *prtdos*=1, together with *iscf* $= -3$, and a metallic *occopt*, between 3 and 7, providing the needed smearing. If *prtdos*=1, the name of the DOS file is the root name for the output files, followed by "_DOS".

If *prtdos*=2, the DOS is computed using the tetrahedron method. As in the case of *prtdos*=1, all levels that are present in the calculation are taken into account (occupied and unoccupied). In this case, the k–points must have been defined using the input variable *ngkpt* or the input variable *kptrlatt*. There must be at least two non–equivalent points in the Irreducible Brillouin Zone to use *prtdos*=2. There is no need to take care of the *occopt* or *tsmear* input variables, and there is no subtlety to be taken into account for insulators. The computation can be done in the self–consistent case as well as in the non–self–consistent case, using *iscf* $= -3$. This allows to refine the DOS at fixed starting density.

In that case, if *ionmov*==0, the name of the potential file will be the root output name, followed by _DOS (like in the *prtdos*=1 case).

However, if *ionmov*==1 or 2, potential files will be output at each time step, with the name being made of

- the root output name,
- followed by _TIMx, where x is related to the timestep (see later)
- then followed by _DOS.

If *prtdos*=3, the same tetrahedron method as for *prtdos*=2 is used, but the DOS inside a sphere centered on some atom is delivered, as well as the angular–momentum projected (l=0,1,2,3,4) DOS in the same sphere. The preparation of this case, the parameters under which the computation is to be done, and the file denomination is similar to the *prtdos*=2 case. However, three additional input variables might be provided, describing the atoms that are the center of the sphere (input variables *natsph* and *iatsph*), as well as the radius of this sphere (input variable *ratsph*).

7.3.24 prteig

Mnemonics: PRinT EIGenenergies
Characteristic:
Variable type: integer parameter
Default is 0 or 1?????.

(Not yet active)

7.3.25 prtfsurf

Mnemonics: PRinT Fermi SURFace file
Characteristic:
Variable type: integer parameter
Default is 0.

If set to 1, print Fermi surface file. For the time being, under development.

7.3.26 prtgeo

Mnemonics: PRinT the GEOmetry analysis
Characteristic:

Variable type: integer parameter
Default is 0.

If set to 1 or a larger value, provide output of geometrical analysis (bond lengths and bond angles). The value of *prtgeo* is taken by the code to be the maximum coordination number of atoms in the system.

It will deduce a maximum number of "nearest" and "next–nearest" neighbors accordingly, and compute corresponding bond lengths.

It will compute bond angles for the "nearest" neighbours only.

If *ionmov*==0, the name of the file will be the root output name, followed by _GEO.

If *ionmov*==1 or 2, one file will be output at each time step, with the name being made of

- the root output name,
- followed by _TIMx, where x is related to the timestep (see later)
- then followed by _GEO

The content of the file should be rather self–explanatory.

No output is provided by *prtgeo* is lower than or equal to 0.

If *prtgeo* > 0, the maximum number of atoms (*natom*) is 9999.

7.3.27 prtkpt

Mnemonics: PRinT the K–PoinTs sets
Characteristic:
Variable type: integer parameter
Default is 0.

If set $\neq 0$, proceeds to a detailed analysis of different k–point grids. Works only if *kptopt* is positive, and neither *kptrlatt* nor *ngkpt* are defined. ABINIT will stop after this analysis.

Different sets of k–point grids are defined, with common values of *shiftk*. In each set, ABINIT increases the length of vectors of the supercell (see *kptrlatt*) by integer steps. The different sets are labelled by "iset". For each k–point grid, *kptrlen* and *nkpt* are computed (the latter always invoking *kptopt*=1, that is, full use of symmetries). A series is finished when the computed *kptrlen* is twice larger than the input variable *kptrlen*. After the examination of the different sets, ABINIT summarizes, for each *nkpt*, the best possible grid, that is, the one with the largest computed *kptrlen*.

Note that this analysis is also performed when *prtkpt*=0, as soon as neither *kptrlatt* nor *ngkpt* are defined. But, in this case, no analysis report is given, and the code selects the grid with the smaller *ngkpt* for the desired *kptrlen*. However, this analysis takes some times (well sometimes, it is only a few seconds — it depends on the value of the input *kptrlen*), and it is better to examine the full analysis for a given cell and set of symmetries, *shiftk* for all the production runs.

7.3.28 prtpot

Mnemonics: PRinT the iotal (kohn–sham)POTential

7.3.29 prtvha

Mnemonics: PRinT V_HArtree

7.3.30 prtvhxc

Mnemonics: PRinT V_(Hartree+XC)

7.3.31 prtvxc

Mnemonics: PRinT V_XC
Characteristic:
Variable type: integer parameter
Default is 0.

If set ≥ 1, provide output of different potentials.

For *prtpot*, output the total (Kohn–Sham) potential, sum of local pseudo–potential, Hartree potential, and xc potential.

For *prtvha*, output the Hartree potential.

For *prtvhxc*, output the sum of Hartree potential and xc potential.

For *prtvxc*, output the exchange–correlation potential.

If *ionmov*==0, the name of the potential file will be the root output name, followed by _POT, _VHA, _VHXC, or _VXC.

If *ionmov*==1 or 2, potential files will be output at each time step, with the name being made of

- the root output name,
- followed by _TIMx, where x is related to the timestep (see later)
- then followed by _POT, _VHA, _VHXC, or _VXC.

The file structure of this unformatted output file is described in section 6.6 of `abinis_help`. No output is provided by a negative value of these variables.

7.3.32 prtstm

Mnemonics: PRinT the STM density
Characteristic:
Variable type: integer parameter
Default is 0.

If set to 1 or a larger value, provide output of the electron density in real space rho(r), made only from the electrons close to the Fermi energy, in a range of energy (positive or negative), determined by the (positive or negative, but non–zero) value of the STM bias *stmbias*.

This is a very approximate way to obtain STM profiles: one can choose an equidensity surface, and consider that the STM tip will follow this surface. Such equidensity surface might be determined with the help of Cut3D, and further post–processing of it (to be implemented). The big approximations of this technique are: neglect of the finite size of the tip, and position–independent transfer matrix elements between the tip and the surface.

The charge density is provided in units of electrons/Bohr^3. The name of the STM density file will be the root output name, followed by _STM. Like a _DEN file, it can be analyzed by cut3d. The file structure of this unformatted output file is described in section 6.5 of `abinis_help`.

For the STM charge density to be generated, one must give, as an input file, the converged wavefunctions obtained from a previous run, at exactly the same k–points and cut–off energy, self–consistently determined, using the occupation numbers from *occopt*=7.

In the run with positive *prtstm*, one has to use:

- positive *iscf*

- *occopt*=7, with specification of *tsmear*
- *nstep*=1
- the *tolwfr* convergence criterion
- *ionmov*=0 (this is the default value)
- *optdriver*=0 (this is the default value)

Note that you might have to adjust the value of *nband* as well, for the treatment of unoccupied states, because the automatic determination of *nband* will often not include enough unoccupied states.

When *prtstm* is non–zero, the stress tensor is set to zero.

No output of _STM file is provided by *prtstm* lower or equal to 0.

7.3.33 prtvol

Mnemonics: PRinT VOLume
Characteristic:
Variable type: integer parameter
Default is 0.

Control the volume of printed output.

Standard choice is 0. Positive values print more in the output and log files, while negative values are for debugging (or preprocessing only), and cause the code to stop at some point.

- 0 ⇒ there is a limit on the number of k–points for which related information will be written. This limit is presently 50. Due to some subtlety, if for some dataset *prtvol* is non–zero, the limit for input and output echoes cannot be enforced, so it is like if *prtvol*=1 for the all the dataset for which *prtvol* was set to 0.
- 1 ⇒ there is no such limit for the input and output echoes, in the main output file.
- 2 ⇒ there is no such limit in the whole main output file.
- 3 ⇒ there is no such limit in both output and log files.
- 10 ⇒ no limit on the number of k–points, and moreover, the eigenvalues are printed for every SCF iteration, as well as other additions (to be specified in the future ...)

Debugging options:

- $= -1$ ⇒ stop in abinis (main program), before call gstate. Useful to see the effect of the preprocessing of input variables (memory needed, effect of symmetries, k–points ...) without going further. Run very fast, on the order of the second.
- $= -3$ ⇒ stop in gstate, before call scfcv, move or brdmin. Useful to debug pseudopotentials
- $= -4$ ⇒ stop in move, after completion of all loops
- $= -5$ ⇒ stop in brdmin, after completion of all loops
- $= -6$ ⇒ stop in scfcv, after completion of all loops
- $= -7$ ⇒ stop in vtorho, after the first rho is obtained
- $= -8$ ⇒ stop in vtowfk, after the first k–point is treated
- $= -9$ ⇒ stop in cgwf, after the first wf is optimized
- $= -10$ ⇒ stop in getghc, after the Hamiltonian is applied once

This debugging feature is not yet activated in the RF routines. Note that *fftalg* offers another option for debugging.

7.3.34 prtwf

Mnemonics: PRinT the WaveFunction
Characteristic:
Variable type: integer parameter
Default is 1.

If set ≥ 1, provide output of wavefunction and eigenvalue file, as described in section 6.7 of the main abinis help file.

For a standard ground–state calculation, the name of the wavefunction file will be the root output name, followed by _WFK. If *nqpt*=1, the root name will be followed by _WFQ. For response–function calculations, the root name will be followed by _1WFx, where x is the number of the perturbation. The dataset information will be added as well, if relevant.

No wavefunction output is provided by *prtwf*=0.

7.3.35 prt1dm

Mnemonics: PRinT 1-DiMensional potential and density
Characteristic:
Variable type: integer parameter
Default is 0.

If set ≥ 1, provide one–dimensional projection of potential and density, for each of the three axis. This corresponds to averaging the potential or the density on bi–dimensional slices of the FFT grid.

7.4 Geometry builder + symmetry related variables, VARGEO

7.4.1 brvltt

Mnemonics: BRaVais LaTTice type
Characteristic: SYMMETRIZER
Variable type: integer parameter
Default is 0.

Set the type of Bravais lattice, needed only if *spgroup* $\neq 0$. In this case, the cell defined by *acell* and *rprim* or *angdeg* should be the CONVENTIONAL cell.

If *brvltt*=0, the code will assign *brvltt* from the space group information *spgroup*, and produce the symmetry operations for the conventional unit cell. If the conventional cell is not primitive, the user should set *chkprim*=0.

If *brvltt* $= -1$, the code will assign *brvltt* from the space group information, then reduce the unit cell to a primitive unit cell. The echo of *acell* and *rprim* might thus differ from those derived directly from the input variables. Also, the input variable *xred* will refer to the CONVENTIONAL unit cell, but its echo will refer to the preprocessed PRIMITIVE unit cell. There is of course no problem with *xangst* and *xcart*, as they are independent of the unit cell.

The echo of *brvltt* in the output file will be one of the following Bravais lattices:

- 1 = Primitive with no associated translations
- 2 = Inner centered with (a/2 + b/2 + c/2) associated translation
- 3 = Face centered with (a/2 + b/2; b/2 + c/2; c/2 + a/2) associated translations
- 4 = C — centered with (a/2 + b/2) associated translation

- 5 = A — centered with (b/2 + c/2) associated translation
- 6 = B — centered with (c/2 + a/2) associated translation
- 7 = Rhombohedral lattice.

The user might also input directly these values, although they might not be consistent with *spgroup*.

The space groups 146, 148, 155, 160, 161, 166, 167, when used with *spgaxor*=1 (hexagonal axes) will have *brvltt*=7 and two associated translations: (2/3, 1/3, 1/3) and (1/3, 2/3, 2/3).

For more details see the space group help file.

7.4.2 genafm

Mnemonics: GENerator of the translation for Anti–FerroMagnetic space group
Characteristic: SYMMETRISER
Variable type: real genafm(3)
Default 3*0.

This input variable might be used to define a Shubnikov type IV magnetic space group (anti–ferromagnetic space group). The user is advised to consult "The mathematical theory of symmetry in solids, Representation theory for point groups and space groups, 1972, C.J. Bradley and A.P. Cracknell, Clarendon Press, Oxford."

A Shubnikov type IV magnetic space group might be defined by its Fedorov space group (set of spatial symmetries, that do not change the magnatisation), and one translation associated with a change of magnetisation. *genafm* is precisely this translation, in reduced coordinates (like *xred*)

Thus, one way to specify a Shubnikov IV magnetic space group, is to define both *spgroup* and *genafm*. Alternatively, one might define *spgroup* and *spgroupma*, or define by hand the set of symmetries, using *symrel*, *tnons* and *symafm*.

7.4.3 natrd

Mnemonics: Number of AToms ReaD
Characteristic: GEOMETRY BUILDER, SYMMETRISER
Variable type: integer parameter
Default is *natom*.

Gives the number of atoms to be read from the input file, in the case the geometry builder or the symmetriser is used. In this case, *natrd* is also used to dimension the array *typat*, and the arrays *xred*, *xangst* and *xcart*.

Must take into account the vacancies (see *vacnum* and *vaclst*).

Despite possible vacancies, cannot be bigger than *natom*.

7.4.4 nobj

Mnemonics: Number of OBJects
Characteristic: GEOMETRY BUILDER, NO INTERNAL
Variable type: integer parameter
Default is 0 (no use of the geometry builder).

Gives the number of 'objects' to be used by the geometry builder in order to find the full set of atoms. At present, only one or two objects can be defined, identified as objects 'a' and 'b'.

Related variables for object 'a' are: *objan*, *objaat*, *objarf*, *objatr*, *objaro*, *objaax*. Related variables for object 'b' are: *objbn*, *objbat*, *objbrf*, *objbtr*, *objbro*, *objbax*.

More detailed explanation: when the geometry builder is used (i.e. when *nobj*==1 or *nobj*==2), the code will be given a primitive set of atoms, from which it will have to deduce the full set of atoms.

An object will be specified by the number of atoms it includes (*objan* or *objbn*), and the list of these atoms (*objaat* or *objbat*).

Examples of physical realisation of an object can be a molecule, or a group of atom to be repeated, or a part of a molecule to be rotated. The geometry builder can indeed repeat these objects (*objarf* or *objbrf*), rotate them (*objaro* or *objbro*) with respect to an axis (*objaax* or *objbax*), and translate them (*objatr* or *objbtr*). After having generated a geometry thanks to rotation, translation and repetition of objects, it is possible to remove some atoms, in order to create vacancies (*vacnum* and *vaclst*). The number of atoms in the primitive set, those that will be read from the input file, is specified by the variable *natrd*. It will be always smaller than the final number of atoms, given by the variable *natom*. The code checks whether the primitive number of atoms plus those obtained by the repetition operation is coherent with the variable *natom*, taking into account possible vacancies.

You should look at the other variables for more information. Go to *objan*, for example.

Not present in the dtset array (no internal).

7.4.5 objaat, objbat

Mnemonics: OBJect A: list of AToms, OBJect B: list of AToms
Characteristic: GEOMETRY BUILDER, NO INTERNAL
Variable type: integer arrays objaat(*objan*) and objbat(*objbn*)

Gives the list of atoms in either object a or object b. This list is specified by giving, for each atom, its index in the list of coordinates (*xred*, *xangst* or *xcart*), that also corresponds to a type of atom (given by the array type). These objects can be thought as molecules, or groups of atoms, or parts of molecules, to be repeated, rotated and translated to generate the full set of atoms.

Look at *objarf* and *objbrf* for further explanations. *objaat* MUST be provided if *nobj*==1. *objaat* and *objbat* MUST be provided if *nobj*==2.

Not present in the dtset array (no internal).

7.4.6 objaax, objbax

Mnemonics: OBJect A: AXis, OBJect B: AXis
Characteristic: GEOMETRY BUILDER, NO INTERNAL, LENGTH
Variable type: real arrays objaax(6) and objbax(6)

Gives, for each object, the cartesian coordinates of two points (first point: *objaax*(1:3) or *objbax*(1:3), second point: *objaax*(4:6) or *objbax*(4:6)).

By default, given in bohr atomic units (1 bohr=0.5291772083 Å), although Angstrom can be specified, if preferred, since these variables have the 'LENGTH' characteristics.

The two points define an axis of rotation of the corresponding object.

Note that the rotation of the object is done BEFORE the object is translated.

The sign of the rotation angle is positive if the object is to be rotated clockwise when looking to it along the axis, from point 1 (coordinates 1:3) toward point 2 (coordinates 4:6).

objaat MUST be provided if *nobj*==1 and one component of *objaro* does not vanish.

objaat and *objbat* MUST be provided if *nobj*==2 and one component of *objbro* does not vanish.

Not present in the dtset array (no internal).

7.4.7 objan, objbn

Mnemonics: OBJect A: Number of atoms, OBJect B: Number of atoms
Characteristic: GEOMETRY BUILDER, NO INTERNAL

Variable type: integer parameters

Gives the number of atoms in either object a or object b. The list of atoms is given by the variables *objaat* and *objbat*.

objan MUST be provided if *nobj*==1.

objan and *objbn* MUST be provided if *nobj*==2.

Not present in the dtset array (no internal).

7.4.8 objarf, objbrf

Mnemonics: OBJect A: Repetition Factors, OBJect B: Repetition Factors
Characteristic: GEOMETRY BUILDER, NO INTERNAL
Variable type: integer arrays objarf(3) and objbrf(3)
Default is 1 1 1.

Gives three repetition factors of the objects a or b.

This gives the opportunity to generate a three–dimensional set of repeated objects, although a simple one–dimensional repetition will be easily obtained through the specification of

nrep 1 1 where nrep is the 1D repetition factor.

The initial rotation and translation of the object, as well as the increment of rotation or translation from one object to the next are specified by the variables *objaro* and *objatr*, for object a, and *objbro* and *objbtr*, for object b.

Note that the geometry builder will generate the full set of atoms from the primitive set of atoms using the following order: it will process each atom in the primitive list one by one, determine whether it belongs to either object a or object b, and then repeat it taking into account the proper rotation and translation, with the fastest varying repetition factor being the first, then the second, then the third.

In the final list of atoms, one will first find the atoms generated from atom 1 in the primitive list, then those generated from atom 2 in the primitive list, and so on.

If the geometry builder is only used to rotate or translate an object, without repeating it, simply use 1 1 1, which is also the Default value.

Not present in the dtset array (no internal).

7.4.9 objaro, objbro

Mnemonics: OBJect A: ROtations, OBJect B: ROtations
Characteristic: GEOMETRY BUILDER, NO INTERNAL
Variable type: real arrays objaro(4) and objbro(4)
Default is 4*0.0d0 (no rotation).

Give, for each object, the angles of rotation in degrees to be applied to the corresponding object.

The rotation is applied before the translation, and the axis is defined by the variables *objaax* and *objbax*. See the latter variables for the definition of the sign of the rotation.

The first component *objaro*(1) and *objbro*(1) gives the angle of rotation to be applied to the first instance of the object. The second, third or fourth component (resp.) gives the increment of rotation angle from one instance to the next instance, defined by the first, second or third repetition factor (resp.). This allows to generate 3D arrays of molecules with different rotation angles.

Not present in the dtset array (no internal).

7.4.10 objatr, objbtr

Mnemonics: OBJect A: TRanslations, OBJect B: TRanslations
Characteristic: GEOMETRY BUILDER, NO INTERNAL, LENGTH
Variable type: real arrays objatr(3,4) and objbtr(3,4)
Default is 12*0.0d0 (no translation).

Give, for each object, the vectors of translations, in cartesian coordinates, to be applied to the corresponding object. By default, given in bohr atomic units (1 bohr=0.5291772083 Å), although Angstrom can be specified, if preferred, since these variables have the 'LENGTH' characteristics.

The translation is applied after the rotation.

The first vector *objatr*(3,1) and *objbro*(3,1) gives the translation to be applied to the first instance of the object. The second, third or fourth component (resp.) gives the increment of translation from one instance to the next instance, defined by the first, second or third repetition factor (resp.). This allows to generate 3D arrays of molecules.

In general, when the objects are repeated, a translation vector must be given, since otherwise, the repeated objects pack in the same region of space. As an exception, one can have a set of molecules regularly spaced on a circle, in which case, only rotations are needed.

Not present in the dtset array (no internal).

7.4.11 ptgroupma

Mnemonics: PoinT GROUP number for the MAgnetic space group
Characteristic: SYMMETRISER, INTERNAL
Variable type: integer parameter
Default 0.

This internal variable characterizes a Shubnikov type III magnetic space group (anti–ferromagnetic space group). The user is advised to consult "The mathematical theory of symmetry in solids, Representation theory for point groups and space groups, 1972, C.J. Bradley and A.P. Cracknell, Clarendon Press, Oxford."

A Shubnikov type III magnetic space group might be defined by its Fedorov space group (set of all spatial symmetries, irrespective of their magnetic action), and the halving space group (only the symmetries that do not change the magnetisation).

The specification of the halving space group might be done by specifying, for each point symmetry, the magnetic action. See Table 7.1 of the above–mentioned reference. Magnetic space groups are numbered from 1 to 58.

7.4.12 spgaxor

Mnemonics: SPace Group: AXes ORientation
Characteristic: SYMMETRISER
Variable type: integer parameter
Default 1.

It is taken into account only when *spgroup* $\neq 0$; it allows one to define the axes orientation for the specific space groups for which this is needed. Trigonal groups (number 146,148,155,160,161,166,167):

- 1 represents the hexagonal axes
- 2 represents the rhombohedral axes

Orthorhombic space groups: there are six possibilities corresponding to the possible axes permutations

- 1 abc → abc

- 2 abc → cab
- 3 abc → bca
- 4 abc → acb
- 5 abc → bac
- 6 abc → cba

Monoclinic: there are 3 or 9 possibilities depending on the space group. See the space group help file for details. In the log/output file the notation used to describe the monoclinic groups is for example:

```
15:c1, A2/a_c = C2/c
```

where,

- `15` represents the space group number,
- `c1` the orientation as it appears on the web page,
- `A` is the real Bravais type lattice,
- `2/a` the existent symmetry elements,
- `_c` marks the orientation of the two–fold axis or of the mirror plane,
- `C2/c` represents the parent space group.

7.4.13 spgorig

Mnemonics: SPace Group: ORIGin
Characteristic: SYMMETRISER
Variable type: integer parameter
Default 1.

Gives the choice of origin for the axes system, taken into account only when *spgroup* $\neq 0$,
It is defined according to the origin choice in the International Tables of Crystallography.
It applies only to the space groups 48, 50, 59, 70, 85, 86, 88, 125, 126, 129, 130, 133, 134, 137, 141, 142, 201, 203, 222, 224, 227, 228.
For details see the space group help file.

7.4.14 spgroup

Mnemonics: SPace GROUP number
Characteristic: SYMMETRISER
Variable type: integer parameter
Default 0.

Gives the number of the space group.
If *spgroup* is 0, the code assumes that all the symmetries are input through the *symrel* matrices and the *tnons* vectors, or obtained from the symmetry finder (the default when *nsym*==0).
It should be between 1 and 230. This option can be used to obtain all the atoms in the unit cell, starting from the assymetric unit cell.
The references for computing the symmetry corresponding to the space groups are:

- International Tables for Crystallography, 1983, Ed. Theo Hahn, D. Reidel Publishing Company

- The mathematical theory of symmetry in solids, Representation theory for point groups and space groups, 1972, C. J. Bradley and A.P. Cracknell, Clarendon Press, Oxford.

For details see the space group help file.

7.4.15 spgroupma

Mnemonics: SPace GROUP number defining a MAgnetic space group
Characteristic: SYMMETRISER, NOT INTERNAL
Variable type: integer parameter
Default 0.

This input variable might be used to define a Shubnikov magnetic space group (anti–ferromagnetic space group). The user is advised to consult "The mathematical theory of symmetry in solids, Representation theory for point groups and space groups, 1972, C. J. Bradley and A. P. Cracknell, Clarendon Press, Oxford."

A Shubnikov type IV magnetic space group might be defined by its Fedorov space group (set of spatial symmetries that do not change the magnetisation), and an additional magnetic space group number *spgroupma*.

A Shubnikov type III magnetic space group might be defined by its Fedorov space group (set of all spatial symmetries, irrespective of their magnetic action), and an additional magnetic space group number *spgroupma*.

For the additional number *spgroupma*, we follow the definition of Table 7.4 of the above–mentioned Bradley and Cracknell textbook.

Thus, one way to specify a Shubnikov IV magnetic space group, is to define both *spgroup* and *spgroupma*.

For example, the group `P2_1/c'` has *spgroup*=14 and *spgroupma*=78.

Alternatively, for Shubnikov IV magnetic groups, one might define *spgroup* and *genafm*. For both the type III and IV, one might define by hand the set of symmetries, using *symrel*, *tnons* and *symafm*

7.4.16 vaclst

Mnemonics: VACancies LiST
Characteristic: GEOMETRY BUILDER, NOT INTERNAL
Variable type: integer array vaclst(*vacnum*)
No Default.

Gives the identification number(s) of atoms to be subtracted from the set of atoms that are obtained after having rotated, translated and repeated the objects.

Useful to created vacancies.

7.4.17 vacnum

Mnemonics: VACancies NUMber
Characteristic: GEOMETRY BUILDER
Variable type: integer parameter
Default value is 0.

Gives the number of atoms to be subtracted from the list of atoms after the rotations, translations and repetitions have been done. The list of these atoms is contained in *vaclst*.

7.5 Ground–state calculation variables, VARGS

7.5.1 algalch

Mnemonics: ALGorithm for generating ALCHemical pseudopotentials
Characteristic:
Variable type: integer array algalch(*ntypalch*)
Default is 1 for all indices

Used for the generation of alchemical pseudopotentials, that is, when *ntypalch* is non–zero.
Give the algorithm to be used to generate the *ntypalch* alchemical potentials from the different *npspalch* pseudopotentials dedicated to this use.
Presently, *algalch* can only have the value 1, that is:

- the local potentials are mixed, thanks to the *mixalch* mixing coefficients
- the form factors of the non–local projectors are all preserved, and all considered to generate the alchemical potential
- the scalar coefficients of the non–local projectors are multiplied by the proportion of the corresponding type of atom that is present in *mixalch*
- the characteristic radius for the core charge is a linear combination of the characteristic radii of the core charges, build with the *mixalch* mixing coefficients
- the core charge function f(r/rc) is a linear combination of the core charge functions, build with the *mixalch* mixing coefficients

Later, other algorithms for the mixing might be included.

7.5.2 bdberry

Mnemonics: BanD limits for BERRY phase
Characteristic:
Variable type: integer array bdberry(4)
Default is 4*0.

Used for non–zero values of *berryopt*.
Give the lower band and the upper band of the set of bands for which the Berry phase must be computed. Irrelevant if *nberry* is not positive. When *nsppol* is 1 (no spin–polarisation), only the two first numbers, giving the lower and highest bands, are significant. Their occupation number is assumed to be 2. When *nsppol* is 2 (spin–polarized calculation), the two first numbers give the lowest and highest bands for spin up, and the third and fourth numbers give the lowest and highest bands for spin down. Their occupation number is assumed to be 1.
Presently, bdband MUST be initialized by the user in case of Berry phase calculation: the above–mentioned default will cause an early exit.

7.5.3 berryopt

Mnemonics: BERRY phase options
Characteristic:
Variable type: integer berryopt
Default is 0

- 0 ⇒ no computation of expressions relying on a Berry phase (default)

- 1 ⇒ the computation of Berry phases is activated (berryphase routine)
- 2 ⇒ the computation of derivatives with respect to the wavevector, thanks to the Berry phase finite–difference formula, is activated (uderiv routine)
- 3 ⇒ same as option 1 and 2 together
- 4 ⇒ finite electric field calculation
- −1 ⇒ alternative computation of Berry phases (berryphase_new routine)
- −2 ⇒ alternative computation of derivatives with respect to the wavevector, thanks to the Berry phase finite–difference formula (berryphase_new routine)
- −3 ⇒ same as option −1 and −2 together

The other related input variables are:

- in case of *berryopt*=1,2, or 3: *bdberry* and *kberry*; also, *nberry* must be larger than 0;
- in case of *berryopt* = −1, −2, or −3: the variable *rfdir* must be used to specify the primitive vector along which the projection of the polarization or the ddk will be computed. For example if *berryopt*=1 and *rfdir*=1 0 0, the projection of the polarization along the reciprocal lattice vector G_1 is computed. In case *rfdir*=1 1 1, ABINIT computes the projection of P along G_1, G_2 and G_3 and transforms the results to cartesian coordinates;
- *efield*, *rfdir* in case of *berryopt*=4;

The cases *berryopt* = −1, −2, −3 and 4 work only if *kptopt*=3, *nsppol*=1, *nspinor*=1, and *occopt*=1.

7.5.4 boxcenter

Mnemonics: BOX CENTER
Characteristic:
Variable type: real array boxcenter(3)
Default boxcenter(1:3) is 0.5 0.5 0.5.

Defines the center of the box, in reduced coordinates. At present, this information is only used in the case of Time–Dependent DFT computation of the oscillator strength. One must take *boxcenter* such as to be roughly the center of the cluster or molecule. The default is sensible when the *vacuum* surrounding the cluster or molecule has *xred* 0 or 1. On the contrary, when the cluster or molecule is close to the origin, it is better to take *boxcenter*=(0 0 0).

7.5.5 boxcutmin

Mnemonics: BOX CUT–off MINimum
Characteristic:
Variable type: real
Default is 2.0.

The box cut–off ratio is the ratio between the wavefunction plane wave sphere radius, and the radius of the sphere that can be inserted in the FFT box, in reciprocal space. In order for the density to be exact (in the case of plane wave, not PAW), this ratio should be at least two. If one uses a smaller ratio, one will gain speed, at the expense of accuracy. In case of pure ground state calculation (e.g. for the determination of geometries), this is sensible. However, the wavefunctions that are obtained CANNOT be used for starting response function calculation.

7.5.6 charge

Mnemonics: CHARGE
Characteristic:
Variable type: real number
Default is 0.

Used to establish charge balance between the number of electrons filling the bands and the nominal charge associated with the atomic cores.

The code adds up the number of valence electrons provided by the pseudopotentials of each type (call this "zval"), then add charge, to get the number of electrons per unit cell, *nelect*.

Then, if *iscf* is positive, the code adds up the band occupancies (given in array *occ*) for all bands at each k–point, then multiplies by the k–point weight *wtk* at each k–point. Call this sum "nelect_occ" (for the number of electrons from occupation numbers). It is then required that: nelect_occ = *nelect*

To treat a neutral system, which is desired in nearly all cases, one must use *charge*=0. To treat a system missing one electron per unit cell, set *charge*=+1.

7.5.7 chkexit

Mnemonics: CHecK whether the user want to EXIT
Characteristic:
Variable type: integer parameter
Default is 2 for sequential version of ABINIT, 1 for parallel version of ABINIT.

If *chkexit* is 1 or 2, ABINIT will check whether the user wants to interrupt the run (using the keyword "exit" on the top of the input file or creating a file named "abinit.exit": see the end of section 3.2 of `abinis_help`).

If *chkexit*=0, the check is not performed at all

If *chkexit*=1, the check is not performed frequently (after each SCF step)

If *chkexit*=2, the check is performed frequently (after a few bands, at each k–point)

7.5.8 chkprim

Mnemonics: CHecK whether the cell is PRIMitive
Characteristic: SYMMETRY FINDER
Variable type: integer parameter
Default is 1.

If the symmetry finder is used (see *nsym*), a non–zero value of *chkprim* will make the code stop if a non–primitive cell is used. If *chkprim*=0, a warning is issued, but the run does not stop.

If you are generating the atomic and cell geometry using *spgroup*, you might generate a PRIMITIVE cell using brvlatt=−1.

7.5.9 cpus, cpum, cpuh

Mnemonics: CPU time limit in: Seconds, Minutes, Hours
Characteristic: NO MULTI; for cpum and cpuh: NO INTERNAL
Variable type: real parameters
Default is 0.0d0.

One of these three real parameters can be defined in the input file, to set up a CPU time limit. When the job reaches that limit, it will try to end smoothly. However, note that this might still take some time. If the user want a firm CPU time limit, the present parameter must be

reduced sufficiently. Intuition about the actual margin to be taken into account should come with experience ...

Note that only one of these three parameters can be defined in a single input file. A zero value has no action of the job.

Internally, only *cpus* is used in the dtset array: adequate conversion factors are used to generate it from *cpum* or *cpuh*.

7.5.10 diecut

Mnemonics: DIelectric matrix Energy CUToff
Characteristic: DEVELOP, ENERGY
Variable type: real parameter
Default diecut is 2.2d0 Ha.

Kinetic energy cutoff that controls the number of planewaves used to represent the dielectric matrix: $(1/2)[(2\pi) * (G_{max})]^2 = ecut$ for G_{max}.

Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV)

All planewaves inside this "basis sphere" centered at G=0 are included in the basis. This is useful only when *iprcel* $\geq$ 21, which means that a preconditioning scheme based on the dielectric matrix is used.

NOTE: a negative *diecut* will define the same dielectric basis sphere as the corresponding positive value, but the FFT grid will be identical to the one used for the wavefunctions. The much smaller FFT grid, used when *diecut* is positive, gives exactly the same results.

No meaning for RF calculations yet.

7.5.11 diegap

Mnemonics: DIelectric matrix GAP
Characteristic: DEVELOP, ENERGY
Variable type: real parameter
Default diegap is 0.1 Ha.

Gives a rough estimation of the dielectric gap between the highest energy level computed in the run, and the set of bands not represented. Used to extrapolate dielectric matrix when *iprcel* $\geq$ 21.

Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV)

No meaning for RF calculations yet.

7.5.12 dielam

Mnemonics: DIelectric matrix LAMbda
Characteristic: DEVELOP
Variable type: real parameter between 0 and 1
Default dielam is 0.5.

Gives the amount of occupied states with mean energy given by the highest level computed in the run, included in the extrapolation of the dielectric matrix. Used when *iprcel* $\geq$ 21.

No meaning for RF calculations yet.

7.5.13 dielng

Mnemonics: model DIElectric screening LeNGth
Characteristic:
Variable type: real parameter
Default is 1.0774841d0 (bohr), for historical reasons.

Used for screening length (in bohr) of the model dielectric function, diagonal in reciprocal space. By default, given in bohr atomic units (1 bohr=0.5291772083 Å), although Angstrom can be specified, if preferred, since *dielng* has the 'LENGTH' characteristics.

This model dielectric function is as follows:

$$\mathrm{diel}(K) = \frac{1 + \mathrm{dielng}^2 K^2}{(1/\mathrm{diemac} + \mathrm{dielng}^2 K^2)\mathrm{diemix}}$$

The inverse of this model dielectric function will be applied to the residual, to give the preconditioned change of potential. Right at $K = 0$, diel(K) is imposed to be 1.

If the preconditioning were perfect, the change of potential would lead to an exceedingly fast solution of the self–consistency problem (two or three steps). The present model dielectric function is excellent for rather homogeneous unit cells.

When K→0, it tends to the macroscopic dielectric constant, eventually divided by the mixing factor *diemix*.

For metals, simply put *diemac* to a very large value (10^6 is OK)

The screening length *dielng* governs the length scale to go from the macroscopic regime to the microscopic regime, where it is known that the dielectric function should tend to 1. It is on the order of 1 bohr for metals with medium density of states at the Fermi level, like Molybdenum, and for Silicon. For metals with a larger DOS at the Fermi level (like Iron), the screening will be more effective, so that *dielng* has to be decreased by a factor of 2-4.

This works for GS and RF calculation.

7.5.14 diemac

Mnemonics: model DIElectric MACroscopic constant
Characteristic:
Variable type: real parameter
Default is 106 (metallic damping).

A rough knowledge of the macroscopic dielectric constant *diemac* of the system is a useful help to speed–up the SCF procedure: a model dielectric function, see the keyword *dielng*, is used for that purpose. It is especially useful for speeding up the treatment of rather homogeneous unit cells.

Some hint: The value of *diemac* should usually be bigger than 1.0d0, on physical grounds.

For metals, simply put *diemac* to a very large value (the default 106 is OK)

For silicon, use 12.0. A similar value is likely to work well for other semiconductors

For wider gap insulators, use 2.0 ... 4.0

For molecules in an otherwise empty big box, try 1.5 ... 3.0

Systems that combine a highly polarisable part and some *vacuum* are rather badly treated by the present version of ABINIT. You have to experiment a bit to find the best *diemac*. If you let *diemac* to its default value, you might even never obtain the self–consistent convergence!

For response function calculations, use the same values as for GS. The improvement in speed can be considerable for small (but non–zero) values of the wavevector.

7.5.15 diemix

Mnemonics: model DIElectric MIXing factor
Characteristic:

Variable type: real parameter
Default is 1.0.

Gives overall factor of the preconditioned residual potential to be transferred in the SCF cycle. It should be between 0.0 and 1.0.

If the model dielectric function were perfect, *diemix* should be 1.0. By contrast, if the model dielectric function does nothing (when *diemac*=1.0d0 or *dielng* is larger than the size of the cell), *diemix* can be used to damp the amplifying factor inherent to the SCF loop.

For molecules, a value on the order 0.5 or 0.33 is rather usual.

When *iscf*=3 or *iscf*=5, *diemix* is only important at the few first iterations when anharmonic effects are important, since these schemes compute their own mixing factor for self–consistency.

7.5.16 dosdeltae

Mnemonics: DOS Delta in Energy
Characteristic: ENERGY
Variable type: real parameter
Default is 0.0.

Defines the linear grid resolution (energy increment) to be used for the computation of the Density–Of–States, when *prtdos* is non–zero.

If *dosdeltae* is set to zero (the default value), the actual increment is 0.001 Ha if *prtdos*=1, and the much smaller value 0.00005 Ha if *prtdos*=2. This different default value arises because the *prtdos*=1 case, based on a smearing technique, gives a quite smooth DOS, while the DOS from the tetrahedron method, *prtdos*=2, is rapidly varying.

7.5.17 efield

Mnemonics: Electric FIELD
Characteristic:
Variable type: real array efield(3)
Default is 3*0.0.

In case *berryopt*=4, a finite electric field calculation is performed. The value of this electric field, and its direction is determined by *efield*. It must be given in atomic units (1 a.u. of electric field= 514220624373.482 V/m, see note below), in cartesian coordinates.

References for the calculation under electric field (based on multi k–point Berry phase):

- Nunes and Vanderbilt, PRL 73, 712 (1994): real–space version of the finite–field Hamiltonian;
- Nunes and Gonze, PRB 63, 155197 (2001): reciprocal–space version of the finite–field Hamiltonian (the one presently implemented), and extensive theoretical analysis
- Souza, Iniguez and Vanderbilt, PRL 89, 117602 (2003): implementation of the finite–field Hamiltonian (reciprocal–space version)

See also Umari, Pasquarello, PRL 90, 027401 (2003).

The atomic unit of electric field strength is: $e_{Cb}/(4\pi\epsilon_0 a_0^2)$, where e_{Cb} is the electronic charge in Coulomb (1.602176462e-19), ϵ_0 is the electric constant (8.854187817d-12 F/m), and a_0 is the Bohr radius in meter (0.5291772083e-10).

7.5.18 enunit

Mnemonics: ENergy UNITs
Characteristic:
Variable type: integer parameter

Default is 0 (eigenvalues in hartree and phonon frequencies in Hartree and cm-1).

Governs the units to be used for output of eigenvalues (and eventual phonon frequencies)

- 0 ⇒ print eigenvalues in hartree;
- 1 ⇒ print eigenvalues in eV;
- 2 ⇒ print eigenvalues in both hartree and eV.

If phonon frequencies are to computed:

- 0⇒ phonon frequencies in Hartree and cm-1;
- 1⇒ phonon frequencies in eV and THz;
- 2⇒ phonon frequencies in hartree, eV, cm-1, Thz and Kelvin.

7.5.19 fband

Mnemonics: Factor for the number of BANDs
Characteristic: NO INTERNAL
Variable type: real parameter, positive or zero
Default is 0.125 in case *occopt*==1 (insulating case) and 0.500 for other values of *occopt* (metallic case). Not used in case *occopt*==0 or 2.

Governs the number of bands to be used in the code in the case the parameter *nband* is not defined in the input file (which means that *occopt* is not equal to 0 or 2).

In case *fband* is 0.0d0, the code computes from the pseudopotential files and the geometry data contained in the input file, the number of electrons present in the system. Then, it computes the minimum number of bands that can accomodate them, and use that value for *nband*.

In case *fband* differs from zero, other bands will be added, just larger than *fband* times the number of atoms. This parameter is not echoed in the top of the main output file, but only the parameter *nband* that it allowed to compute. It is also not present in the dtset array (no internal).

The default values are chosen such as to give naturally some conduction bands. This improves the robustness of the code, since this allows to identify lack of convergence coming from (near-)degeneracies at the Fermi level. In the metallic case, the number of bands generated might be too small if the smearing factor is large. The occupation numbers of the higher bands should be small enough such as to neglect higher bands. It is difficult to automate this, so a fixed default value has been chosen.

7.5.20 fixmom

Mnemonics: FIX the magnetic MOMent
Characteristic:
Variable type: real parameter
Default is −99.99

This input variable is active only in the *nsppol*=2 case. If *fixmom* is not the "magic" value of −99.99, the magnetic moment of the system will be fixed to the value of *fixmom*. Otherwise, the magnetic moment will be determined self–consistently, by having the same spin up and spin down Fermi energy.

Note: for the time being, only the spin down Fermi energy is written out in the main output file. In the fixed magnetic moment case, it differs from the spin up Fermi energy.

7.5.21 iatsph

Mnemonics: Index for the ATomic SPHeres of the atom–projected density–of–states
Characteristic:
Variable type: integer array iatsph(1:*natsph*)
Default is 1, 2, ... *natsph*

This input variable is active only in the *prtdos*=3 case.

It gives the number of the *natsph* atoms around which the sphere for atom–projected density–of–states will be build, in the *prtdos*=3 case. The radius of these spheres is given by *ratsph*.

7.5.22 iprcel

Mnemonics: Integer for PReConditioning of ELectron response
Characteristic:
Variable type: integer parameter
Default is 0.

Used when *iscf*>0, to define the SCF preconditioning scheme. Potential–based preconditioning schemes for the SCF loop (electronic part) are STILL UNDER DEVELOPMENT. The present parameter (electronic part) describe the way the change of potential is derived from the residual.

The possible values of *iprcel* correspond to:

- 0 ⇒ model dielectric function described by *diemac*, *dielng* and *diemix*.
- larger or equal to 21 ⇒ will compute the dielectric matrix according to *diecut*, *dielam*, *diegap*.
- Between 21 and 29 ⇒ for the first few steps uses the same as option 0 then compute RPA dielectric function, and use it as such.
- Between 31 and 39 ⇒ for the first few steps uses the same as option 0 then compute RPA dielectric function, and use it, with the mixing factor *diemix*.
- Between 41 and 49 ⇒ compute the RPA dielectric matrix at the first step, and recompute it at a later step, and take into account the mixing factor *diemix*.
- Between 51 and 59 ⇒ same as between 41 and 49, but compute the RPA dielectric matrix by another mean
- Between 61 and 69 ⇒ same as between 41 and 49, but compute the electronic dielectric matrix instead of the RPA one.

The step at which the dielectric matrix is computed or recomputed is determined by modulo(*iprcel*,10).

For non–homogeneous cells, relatively large, *iprcel*=45 will likely give a large improvement over *iprcel*=0.

For *nsppol*=2 with metallic *occopt*, only *iprcel*=0 is allowed.

No meaning for RF calculations yet.

7.5.23 kberry

Mnemonics: K wavevectors for BERRY phase computation
Characteristic:
Variable type: integer array kberry(3,*nberry*)
Default is an array of 0

Used for non–zero values of *berryopt*.

This array defines, for each Berry phase calculation (the number of such calculations is defined by *nberry*), the difference of wavevector between k–points for which the overlap matrix must be computed. The polarisation vector will be projected on the direction of that wavevector, and the result of the computation will be the magnitude of this projection. Doing more than one wavevector, with different independent direction, allows to find the full polarisation vector. However, note that converged results need oriented grids, denser along the difference wavevector than usual Monkhorst–Pack grids.

The difference of wavevector is computed in the coordinate system defined by the k–points grid (see *ngkpt* and *kptrlatt*), so that the values of *kberry* are integers. Of course, such a k–point grid must exist, and all the corresponding wavefunctions must be available, so that the computation is allowed only when *kptopt* is equal to 3. In order to save computing time, it is suggested to make a preliminary calculation of the wavefunctions on the irreducible part of the grid, with *kptopt* equal to 1, and then use these converged wavefunctions in the entire Brillouin zone, by reading them to initialize the *kptopt*=3 computation.

7.5.24 kptbounds

Mnemonics: K PoinTs BOUNDarieS
Characteristic: NOT INTERNAL
Variable type: real array kptbounds(3,abs(*kptopt*)+1)
No Default

It is used to generate the circuit to be followed by the band structure, when *kptopt* is negative (it is not read if *kptopt* is zero or positive).

There are abs(*kptopt*) segments to be defined, each of which wich start from the end point of the preceeding one. Thus, the number of points to be input is abs(*kptopt*)+1. They form a circuit starting at *kptbounds*(1:3,1)/*kptnrm* and ending at *kptbounds*(1:3,abs(*kptopt*)+1)/*kptnrm*. The number of divisions of each segment is defined by *ndivk*.

7.5.25 kptrlatt

Mnemonics: K - PoinTs grid: Real space LATTice
Characteristic:
Variable type: integer array kptrlatt(3,3)
No default.

This input variable is used only when *kptopt* is positive. It partially defines the k–point grid. The other piece of information is contained in *shiftk*. *kptrlatt* cannot be used together with *ngkpt*.

The values *kptrlatt*(1:3,1), *kptrlatt*(1:3,2), *kptrlatt*(1:3,3) are the coordinates of three vectors in real space, expressed in the *rprim* coordinate system (reduced coordinates). They defines a super–lattice in real space. The k–point lattice is the reciprocal of this super–lattice, eventually shifted (see *shiftk*).

If neither *ngkpt* nor *kptrlatt* are defined, ABINIT will automatically generate a set of k–point grids, and select the best combination of *kptrlatt* and *shiftk* that allows to reach a sufficient value of *kptrlen*. See this latter variable for a complete description of this procedure.

7.5.26 kptrlen

Mnemonics: K - PoinTs grid: Real space LENgth
Characteristic:
Variable type: real parameter
Default 20.0d0.

This input variable is used only when *kptopt* is positive and non–zero.

Preliminary explanation: The k–point lattice defined by *ngkpt* or *kptrlatt* is used to perform integrations of periodic quantities in the Brillouin Zone, like the density or the kinetic energy. One can relate the error made by replacing the continuous integral by a sum over k–point lattice to the Fourier transform of the periodic quantity. Erroneous contributions will appear only for the vectors in real space that belong to the reciprocal of the k–point lattice, except the origin. Moreover, the expected size of these contributions usually decreases exponentially with the distance. So, the length of the smallest of these real space vectors is a measure of the accuracy of the k–point grid.

When either *ngkpt* or *kptrlatt* is defined, *kptrlen* is not used as an input variable, but the length of the smallest vector will be placed in this variable, and echoed in the output file.

On the other hand, when neither *ngkpt* nor *kptrlatt* are defined, ABINIT will automatically generate a large set of possible k–point grids, and select among this set, the grids that give a length of smallest vector LARGER than *kptrlen*, and among these grids, the one that, when used with *kptopt*=1, reduces to the smallest number of k–points. Note that this procedure can be time–consuming. It is worth to do it once for a given unit cell and set of symmetries, but not use this procedure by default. The best is then to set *prtkpt*=1, in order to get a detailed analysis of the set of grids.

If some layer of vacuum is detected in the unit cell (see the input variable *vacuum*), the computation of *kptrlen* will ignore the dimension related to the direction perpendicular to the vacuum layer, and generate a bi–dimensional k–point grid. If the system is confined in a tube, a one–dimensional k–point grid will be generated. For a cluster, this procedure will only generate the Gamma point.

7.5.27 mixalch

Mnemonics: MIXing coefficients for ALCHemical potentials
Characteristic:
Variable type: integer array mixalch(*npspalch*,*ntypalch*)
Default is 0.d0 (will not accepted!)

Used for the generation of alchemical pseudoatoms, that is, when *ntypalch* is non–zero.

This array gives, for each type of alchemical pseudatom (there are *ntypalch* such pseudoatoms), the mixing coefficients of the basic *npspalch* pseudopotentials for alchemical use. For each type of alchemical pseudoatom, the sum of the mixing coefficients must equal 1.

The actual use of the mixing coefficients is defined by the input variable *algalch*.

Example 1. Suppose that we want to describe Ba(0.25) Sr(0.75) Ti O3. The input variables related to the construction of the alchemical Ba(0.25) Sr(0.75) potential will be:

```
npsp   4               ! 4 pseudopotentials should be read.
znucl  8 40 56 38      ! The nuclear charges. Note that the two
                       ! atoms whose pseudopotentials are to be mixed
                       ! are mentioned at the end of the series.
ntypat  3              ! There will be three types of atoms.
ntypalch   1           ! One pseudoatom will be alchemical.
                       ! Hence, there will be ntyppure=2 pure pseudoatoms,
                       ! with znucl 8 (O) and 40 (Ti), corresponding to
                       ! the two first pseudopotentials. Out of the
                       ! four pseudopotentials, npspalch=2 are left
                       ! for alchemical purposes, with znucl 56 (Ba)
                       ! and 38 (Sr).
mixalch    0.25  0.75  ! For that unique pseudoatom to be
                       ! generated, here are the mixing coeeficients,
                       ! to be used to combine the Ba and Sr pseudopotentials.
```

Example 2. More complicated, and illustrate some minor drawback of the design of input variables. Suppose that one wants to generate Al(0.25) Ga(0.75) As(0.10)Sb(0.90). The input

variables will be:

```
npsp  4                  ! 4 pseudopotentials should be read
znucl  13 31 33 51       ! The atomic numbers. All pseudopotentials
                         ! will be used for some alchemical purpose
ntypat  2                ! There will be two types of atoms.
ntypalch   2             ! None of the atoms will be ‘‘pure’’.
                         ! Hence, there will be npspalch=4 pseudopotentials
                         !  to be used for alchemical purposes.
mixalch 0.25  0.75 0.0  0.0   ! This array is a (4,2) array, arranged in the
        0.0   0.0  0.1  0.9   ! usual Fortran order.
```

Minor drawback: one should not forget to fill *mixalch* with the needed zero's, in this later case.

7.5.28 natsph

Mnemonics: Number of ATomic SPHeres for the atom–projected density–of–states
Characteristic:
Variable type: integer parameter
Default is *natom*

This input variable is active only in the *prtdos*=3 case.

It gives the number of atoms around which the sphere for atom–projected density–of–states will be build, in the *prtdos*=3 case. The indices of these atoms is given by *iatsph*. The radius of these spheres is given by *ratsph*.

7.5.29 nbdbuf

Mnemonics: Number of BanDs for the BUFfer
Characteristic:
Variable type: integer parameter
Default 0. However, the default is changed to 2 in some cases, see later.

nbdbuf gives the number of bands, the highest in energy, that, among the *nband* bands, are to be considered as part of a buffer. This concept is useful in two situations: in non–self–consistent calculations, for the determination of the convergence tolerance; for response functions of metals, to avoid instabilities.

In non–self–consistent GS calculations (*iscf* < 0), the highest levels might be difficult to converge, if they are degenerate with another level, that does not belong to the set of bands treated.

Then, it might take extremely long to reach *tolwfr*, although the other bands are already extremely well–converged, and the energy of the highest bands (whose residual are not yet good enough), is also rather well converged.

In response to this problem, for non–zero *nbdbuf*, the largest residual (residm), to be later compared with *tolwfr*, will be computed only in the set of non–buffer bands (this modification applies for non–self–consistent as well as self–consistent calculation, for GS as well as RF calculations).

For a GS calculation, with *iscf*<0, supposing *nbdbuf* is not initialized in the input file, then ABINIT will overcome the default *nbdbuf* value, and automatically set *nbdbuf* to 2.

In metallic RF calculations, in the conjugate gradient optimisation of first–order wavefunctions, there is an instability situation when the q wavevector of the perturbation brings the eigenenergy of the highest treated band at some k–point higher than the lowest untreated eigenenergy at some $k+q$ point. If one accept a buffer of frozen states, this instability can be made to disappear. Frozen states receive automatically a residual value of -0.1d0. For a RF calculation, with $3 \leq$ *occopt* ≤ 7, supposing *nbdbuf* is not initialized in the input file, then ABINIT will overcome the default *nbdbuf* value, and automatically set *nbdbuf* to 2. This value might be too low in some cases.

Also, the number of active bands, in all cases, is imposed to be at least 1, irrespective of the value of *nbdbuf*.

7.5.30 nberry

Mnemonics: Number of BERRY phase computations
Characteristic:
Variable type: integer *nberry*
Default is 1

Used for non–zero values of *berryopt*.

Gives the number of Berry phase computations of polarisation, or finite–difference estimations of the derivative of wavefunctions with respect to the wavevector, each of which might be characterized by a different change of wavevector *kberry*.

When equal to 0, no Berry phase calculation of polarisation is performed. The maximal value of *nberry* is 20.

Note that the computation of the polarisation for a set of bands having different occupation numbers is meaningless (although in the case of spin–polarized calculations, the spin up bands might have an identical occupation number, that might differ from the identical occupation number of spin down bands). Although meaningless, ABINIT will perform such computation, if required by the user. The input variable *bdberry* governs the set of bands for which a Berry phase is computed.

The computation of the Berry phase is not yet implemented for spinor wavefunctions (*nspinor*=2). Moreover, it is not yet implemented in the parallel version of ABINIT.

7.5.31 ndivk

Mnemonics: Number of DIVisions of K lines
Characteristic: NOT INTERNAL
Variable type: integer array ndivk(abs(*kptopt*))
No default.

Gives the number of divisions of each of the segments of the band structure, whose path is determined by *kptopt* and *kptbounds*. This is only needed when *kptopt* is negative. In this case, the absolute value of *kptopt* is the number of such segments.

For example, suppose that the number of segment is just one (*kptopt* $= -1$), a value *ndivk*=4 will lead to the computation of points with relative coordinates 0.0, 0.25, 0.5, 0.75 and 1.0, along the segment in consideration.

Now, suppose that there are two segments (*kptopt*=-2), with *ndivk*(1)=4 and *ndivk*(2)=2, the computation of the eigenvalues will be done at 7 points, 5 belonging to the first segment, with relative coordinates 0.0, 0.25, 0.5, 0.75 and 1.0, the last one being also the starting point of the next segment, for which two other points must be computed, with relative coordinates 0.5 and 1.0.

It is easy to compute disconnected circuits (non–chained segments), by separating the circuits with the value *ndivk*=1 for the intermediate segment connecting the end of one circuit with the beginning of the next one (in which case no intermediate point is computed along this segment).

7.5.32 ngfft

Mnemonics: Number of Grid points for Fast Fourier Transform
Characteristic:
Variable type: integer array ngfft(3)
Default is 0 0 0 (so, automatic selection of optimal values)

Gives the size of fast fourier transform (fft) grid in three dimensions. Each number must be composed of the factors 2, 3, and 5 to be consistent with the radices available in our fft. If no *ngfft* is provided or if *ngfft* is set to 0 0 0, the code will automatically provide an optimal set of *ngfft* values, based on *acell*, *rprim* and *ecut*. This is the recommended procedure, of course.

The total number of FFT points is the product: $ngfft(1) \times ngfft(2) \times ngfft(3) = nfft$.

When *ngfft* is made smaller than recommended values, the code runs faster and the equations in effect are approximated by a low pass fourier filter. The code reports to standard output (unit 06) a parameter "boxcut" which is the smallest ratio of the fft box side to the G vector basis sphere diameter. When boxcut is less than 2 the fourier filter approximation is being used. When boxcut gets less than about 1.5 the approximation may be too severe for realistic results and should be tested against larger values of *ngfft*. When boxcut is larger than 2, *ngfft* could be reduced without loss of accuracy. In this case, the small variations that are observed are solely due to the xc quadrature, that may be handled with *intxc*=1 to even reduce this effect.

Internally, *ngfft* is an array of size 18. The present components are stored in *ngfft*(1:3), while

- *ngfft*(4:6) contains slightly different (larger) values, modified for efficiency of the FFT
- *ngfft*(7) is *fftalg*
- *ngfft*(8) is *fftcache*
- *ngfft*(9) is set to 0 if the parallelization of the FFT is not activated, while it is set to 1 if it is activated.
- *ngfft*(10) is the number of processors of the FFT group
- *ngfft*(11) is the index of the processor in the group of processors
- *ngfft*(12) is n2proc, the number of x–z planes, in reciprocal space, treated by the processor
- *ngfft*(13) is n3proc, the number of x–y planes, in real space, treated by the processor
- *ngfft*(14) is mpi_comm_fft, the handle on the MPI communicator in charge of the FFT parallelisation
- *ngfft*(15:18) are not yet used

The number of points stored by this processor in real space is n1*n2*n3proc, while in reciprocal space, it is n1*n2proc*n3.

7.5.33 nline

Mnemonics: Number of LINE minimisations
Characteristic:
Variable type: integer parameter
Default is 4.

Gives maximum number of line minimizations allowed in preconditioned conjugate gradient minimization for each band. The Default, 4, is fine.

Special cases, with degeneracies or near–degeneracies of levels at the Fermi energy may require a larger value of *nline* (5 or 6?) Line minimizations will be stopped anyway when improvement gets small. With the input variable *nnsclo*, governs the convergence of the wavefunctions for fixed potential.

Note that *nline*=0 can be used to diagonalize the Hamiltonian matrix in the subspace spanned by the input wavefunctions.

7.5.34 npsp

Mnemonics: Number PSeudoPotentials
Characteristic: NO MULTI
Variable type: integer parameter
Default is *ntypat*

Usually, the number of pseudopotentials to be read is equal to the number of type of atoms. However, in the case an alchemical mixing of pseudopotential is to be used, often the number of pseudopotentials to be read will not equal the number of types of atoms.

Alchemical pseudopotentials will be present when *ntypalch* is non–zero. See *ntypalch* to understand how to use alchemical potentials in ABINIT. The input variables *ntypalch*, *algalch*,*mixalch*) are active, and generate alchemical potentials from the available pseudopotentials. Also, the inner variables *ntyppure*,*npspalch*) becomes active. See these input variables, especially *mixalch*, to understand how to use alchemical potentials in ABINIT.

7.5.35 npspalch

Mnemonics: Number of PSeudoPotentials that are "ALCHemical"
Characteristic: Inner
Variable type: integer parameter, non–negative

npspalch = npsp − *ntyppure*.

7.5.36 nqpt

Mnemonics: Number of Q - POINTs
Characteristic:
Variable type: integer parameter
Default is 0.

Determines whether one q–point must be read (See the variables *qpt* and *qptnrm*).

Can be either 0 or 1.

If 1 and used in ground–state calculation, a global shift of all the k–points is applied, to give calculation at $k + q$. In this case, the output wavefunction will be appended by _WFQ instead of _WFK (see the section 4 of `abinis_help`) Also, if 1 and a RF calculation is done, defines the wavevector of the perturbation.

7.5.37 nspden

Mnemonics: Number of SPin–DENsity components
Characteristic: DEVELOP
Variable type: integer parameter
The Default is the value of *nsppol*.

If *nspden*=1, no spin–magnetisation: the density matrix is diagonal, with same values spin–up and spin–down (compatible with *nsppol*=1 only, for both *nspinor*=1 or 2)

If *nspden*=2, scalar magnetization (the axis is arbitrarily fixed in the z direction): the density matrix is diagonal, with different values for spin–up and spin–down (compatible with *nspinor*=1, either with *nsppol*=2 — general collinear magnetisation — or *nsppol*=1 — antiferromagnetism)

If *nspden*=4, vector magnetization: the density matrix is full, with allowed x, y and z magnetisation (useful only with *nspinor*=2 and *nsppol*=1, either because there is spin–orbit without time–reversal symmetry — and thus spontaneous magnetization, or with spin–orbit, if one allows for spontaneous non–collinear magnetism). Not yet available for forces, stresses, response functions.

The default (*nspden* = *nsppol*) does not suit the case of vector magnetization.

7.5.38 nspinor

Mnemonics: Number of SPINORial components of the wavefunctions
Characteristic: DEVELOP
Variable type: integer parameter
The Default is 1.

If *nspinor*=1, usual case: scalar wavefunction (compatible with (*nsppol*=1, *nspden*=1) as well as (*nsppol*=2, *nspden*=2))

If *nspinor*=2, the wavefunction is a spinor (compatible with *nsppol*=1, with *nspden*=1 or 4, but not with *nsppol*=2)

When *nspinor* is 2, the values of *istwfk* are automatically set to 1. Also, the number of bands, for each k–point, should be even.

7.5.39 ntypalch

Mnemonics: Number of TYPe of atoms that are "ALCHemical"
Characteristic:
Variable type: integer parameter
The default is 0

Used for the generation of alchemical pseudopotentials: when *ntypalch* is non–zero, alchemical mixing will be used.

Among the *ntypat* types of atoms, the last *ntypalch* will be "alchemical" pseudoatoms, while only the first *ntyppure* will be uniquely associated with a pseudopotential (the *ntyppure* first of these, actually). The *ntypalch* types of alchemical pseudoatoms are to be made from the remaining *npspalch* pseudopotentials.

In this case, the input variables *algalch*, *mixalch* are active, and generate alchemical potentials from the available pseudopotentials. See these input variables, especially *mixalch*, to understand how to use alchemical potentials in ABINIT.

7.5.40 ntyppure

Mnemonics: Number of TYPe of atoms that are "PURe"
Characteristic: Inner
Variable type: integer parameter, non–negative

$$ntyppure = ntypat - ntypalch.$$

7.5.41 occ

Mnemonics: OCCupation numbers
Characteristic: EVOLVING
Variable type: real array occ(*nband*)
Default: occ is set to 0's.

Gives occupation numbers for all bands in the problem. Needed if *occopt*==0 or *occopt*==2. Ignored otherwise. Also ignored when $iscf = -2$.

Typical band occupancy is either 2 or 0, but can be 1 for half–occupied band or other choices in special circumstances.

If *occopt* is not 2, then the occupancies must be the same for each k–point.

If *occopt*=2, then the band occupancies must be provided explicitly for each band, EACH k–POINT, and EACH SPIN–POLARIZATION, in an array which runs over all bands, k–points, and spin–polarizations.

The order of entries in the array would correspond to all bands at the first k–point (spin up), then all bands at the second k–point (spin up), etc, then all k–points spin down.

The total number of array elements which must be provided is (*nband*(1)+*nband*(2)+ ... + *nband*(*nkpt*)) * *nsppol*.

The occupation numbers evolve only for metallic occupations, that is, *occopt* ≥ 3.

7.5.42 optdriver

Mnemonics: OPTions for the DRIVER
Characteristic:
Variable type: integer parameter
The Default is optdriver=0

For each dataset, choose the task to be done, at the level of the "driver" routine.

The choice is between:

- *optdriver*=0: ground–state calculation (GS), routine "gstate"
- *optdriver*=1: response–function calculation (RF), routine "respfn"
- *optdriver*=2: susceptibility calculation (SUS), routine "suscep"
- *optdriver*=3: susceptibility and dielectric matrix calculation (CHI), routine "screening" (see the input variables *ecutwfn*, *ecuteps*, *plasfrq*, *getkss*, as well as *nbandkss* and *nband*)
- *optdriver*=4: self–energy calculation (SIG), routine "sigma"
- *optdriver*=5: non–linear response functions, using the 2n+1 theorem, routine "nonlinear"

If one of *rfphon*, *rfelfd*, or *rfstrs* is non–zero, while *optdriver* is not defined in the input file, ABINIT will set *optdriver* to 1 automatically. These input variables (*rfphon*, *rfelfd*, and *rfstrs*) must be zero if *optdriver* is not set to 1.

7.5.43 so_typat

Mnemonics: Spin–Orbit: TYPe of each pseudo–ATom

7.5.44 pspso (obsolete)

Mnemonics: PSeudoPotential: treatment of Spin–Orbit interaction
Characteristic:
Variable type: integer array *so_typat*(*ntypat*)
Default is *ntypat**1

For each type of atom (each pseudopotential), specify the spin–orbit interaction.

- If 1: no spin–orbit interaction, even if *nspinor*=2
- If 2: treat spin–orbit in the HGH form (not allowed for all pseudopotentials)
- If 3: treat spin–orbit in the HFN form (not allowed for all pseudopotentials)

Also, *so_typat*=0 default to 1, 2, or 3 according to the data contained in the pseudopotential file (1= there is no spin–orbit information in the psp file; 2= the spin–orbit information is of the HGH form; 3= the spin–orbit information is of the HFN form)

7.5.45 qpt

Mnemonics: Q PoinT
Characteristic:
Variable type: real array of 3 elements
Default wavevector is 0 0 0.

Define a q vector.

See *qptnrm* for extra normalization.

In ground–state calculation, if *nqpt* is 1, the vector *qptn*(1:3)= *qpt*(1:3)/*qptnrm* is added to each renormalized k–point *kpt*(1:3)/*kptnrm* to generate the normalized, shifted, set of k–points *kptns*(1:3,1:*nkpt*).

In response–function calculations, *qptn*(1:3)= *qpt*(1:3)/*qptnrm* is the wavevector of the phonon–type calculation.

For insulators, there is no restriction on the q–points to be used for the perturbations. By contrast, for metals, for the time being, it is adviced to take q–points for which the k and $k+q$ grids are the same (when the periodicity in reciprocal space is taken into account).

Tests remains to be done to see whether other q–points might be allowed (perhaps with some modification of the code).

7.5.46 qptnrm

Mnemonics: Q PoinTs NoRMalization
Characteristic:
Variable type: real parameter
Default is 1.0

Provides re–normalization of *qpt*. Must be positive, non–zero. The actual q vector (renormalized) is *qptn*(1:3)= *qpt*(1:3)/*qptnrm*.

7.5.47 ratsph

Mnemonics: Radius of the ATomic SPHere
Characteristic:
Variable type: real parameter
Default is 2.0 Bohr

Active only in the *prtdos*=3 case, for the time being. Provides the radius of the spheres around the *natsph* atoms of indices *iatsph*, in which the local DOS and its angular–momentum projections will be analysed.

Note that, as presently implemented, the SAME radius is used for all the atoms. So, one might have to perform different calculations to obtain the set of relevant DOS, each corresponding to one atom type, for each of which a different radius might be used.

NOTE: The choice of this radius is quite arbitrary. In a plane–wave basis set, there is no natural definition of an atomic sphere. However, it might be wise to use the following well–defined and physically motivated procedure (in version 4.2, this procedure is NOT implemented, unfortunately): from the Bader analysis, one can define the radius of the sphere that contains the same charge as the Bader volume. This "Equivalent Bader charge atomic radius" might then be used to perform the present analysis. See the AIM (Bader) help file for more explanations. Another physically motivated choice would be to rely on another charge partitioning, like the Hirshfeld one (see the cut3d utility). The advantage of using charge partitioning schemes comes from the fact that the sum of atomic DOS, for all angular momenta and atoms, integrated on the energy range of the occupied states, gives back the total charge. If this is not an issue, one could rely on the half of the nearest–neighbour distances, or any scheme that allows to define an atomic

radius. Note that the choice of this radius is however critical for the balance between the s, p and d components. Indeed, the integrated charge within a given radius, behave as a different power of the radius, for the different channels s, p, d. At the limit of very small radii, the s component dominates the charge contained in the sphere ...

7.5.48 spinat

Mnemonics: SPIN for AToms
Characteristic:
Variable type: real array spinat(3,*natom*) or spinat(3,*natrd*) if the symmetriser is used
Default is 0.0d0.

Gives the initial electronic spin–magnetisation for each atom, in unit of h–bar/2.

Note that if *nspden*=2, the z–component must be given for each atom, in triplets (0 0 z–component).

For example, the electron of an hydrogen atom can be spin up (0 0 1.0) or spin down (0 0 –1.0).

This value is only used to create the first exchange and correlation potential, and is not used anymore afterwards.

It is not checked against the initial occupation numbers *occ* for each spin channel.

It is meant to give an easy way to break the spin symmetry, and to allow to find stable local spin fluctuations, for example: antiferromagnetism, or the spontaneous spatial spin separation of elongated H2 molecule.

If the geometry builder is used, *spinat* will be related to the preprocessed set of atoms, generated by the geometry builder. The user must thus foresee the effect of this geometry builder (see *objarf*).

If the geometry builder is not used, and the symmetries are not specified by the user (*nsym*=0), *spinat* will be used, if present, to determine the anti–ferromagnetic characteristics of the symmetry operations, see *symafm*

If the symmetries are specified, and the irreducible set of atoms is specified, the anti–ferromagnetic characteristics of the symmetry operations *symafm* will be used to generate *spinat* for all the non–irreducible atoms.

7.5.49 stmbias

Mnemonics: Scanning Tunneling Microscopy BIAS voltage
Characteristic: ENERGY
Variable type: real parameter
Default is 0.00

Gives, in Hartree, the bias of the STM tip, with respect to the sample, in order to generate the STM density map.

Used with positive *iscf*, *occopt*=7 (metallic, gaussian), *nstep*=1, and positive *prtstm*, this value is used to generate a charge density map from electrons close to the Fermi energy, in a (positive or negative) energy range. Positive *stmbias* will lead to the inclusion of occupied (valence) states only, while negative *stmbias* will lead to the inclusion of unoccupied (conduction) states only.

Can be specified in Ha (the default), Ry, eV or Kelvin, since *stmbias* has the 'ENERGY' characteristics. 0.001 Ha = 27.2113961 meV = 315.773 Kelvin. With *occopt*=7, one has also to specify an independent broadening *tsmear*.

7.5.50 symafm

Mnemonics: SYMmetries, Anti–FerroMagnetic characteristics
Characteristic:
Variable type: integer array symafm(*nsym*)

Default is *nsym**1.

In case the material is magnetic (well, this is only interesting in the case of antiferromagnetism), additional symmetries might appear, that change the sign of the magnetisation. They have been introduced by Shubnikov (1951). They can be used by ABINIT to decrease the CPU time, by using them to decrease the number of k–points.

symafm should be set to +1 for all the usual symmetry operations, that do not change the sign of the magnetisation, while it should be set to −1 for the magnetisation–changing symmetries.

If the symmetry operations are not specified by the user in the input file, that is, if *nsym*=0, then ABINIT will use the values of *spinat* to determine the content of *symafm*.

7.5.51 timopt

Mnemonics: TIMing OPTion
Characteristic: NO MULTI, DEVELOP
Variable type: integer parameter
Default is 1 for sequential code, 2 for parallel code.

This input variable allows to modulate the use of the timing routines.

- If 0 ⇒ as soon as possible, suppresses all calls to timing routines
- If 1 ⇒ usual timing behaviour, with short analysis, appropriate for sequential execution
- If 2 ⇒ close to *timopt*=1, except that the analysis routine does not time the timer, appropriate for parallel execution.
- If −1 ⇒ a full analysis of timings is delivered
- If −2 ⇒ a full analysis of timings is delivered, except timing the timer

7.5.52 tphysel

Mnemonics: Temperature (PHYSical) of the ELectrons
Characteristic: ENERGY
Variable type: real parameter
Default is 0.00

Gives, in Hartree, the physical temperature of the system, in case *occopt*=4, 5, 6, or 7.

Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. 0.001 Ha = 27.2113961 meV = 315.773 Kelvin. One has to specify an independent broadening *tsmear*. The combination of the two parameters *tphysel* and *tsmear* is described in a paper by M. Verstraete and X. Gonze, Phys. Rev. B (2002). Note that the signification of the entropy is modified with respect to the usual entropy. The choice has been made to use *tsmear* as a prefactor of the entropy, to define the entropy contribution to the free energy.

7.5.53 tsmear

Mnemonics: Temperature of SMEARing
Characteristic: ENERGY
Variable type: real parameter
Default is 0.04

Gives the broadening of occupation numbers *occ*, in the metallic cases (*occopt*=3, 4, 5, 6 and 7). Can be specified in Ha (the default), eV, Ry, or Kelvin, since *tsmear* has the 'ENERGY' characteristics. 0.001 Ha = 27.2113961 meV = 315.773 Kelvin

Default is 0.04 Ha. This should be OK for a free–electron metal like Al. For d–band metals, use 0.01 Ha.

Always check the convergence of the calculation with respect to this parameter, and simultaneously, with respect to the sampling of k–points (see *nkpt*)

If *occopt*=3, *tsmear* is the physical temperature, as the broadening is based on Fermi–Dirac statistics. However, if *occopt*=4, 5, 6, or 7, the broadening is not based on Fermi–Dirac statistics, and *tsmear* is only a convergence parameter. It is still possible to define a physical temperature, thanks to the input variable *tphysel*. See the paper by M. Verstraete and X. Gonze, Phys. Rev. B (2002).

7.5.54 vacuum

Mnemonics: VACUUM identification
Characteristic: NOT INTERNAL
Variable type: integer array vacuum(3)
No Default

Establishes the presence (if 1) or absence (if 0) of a vacuum layer, along the three possible directions normal to the primitive axes.

This information might be used to generate k–point grids, if *kptopt*=0 and neither *ngkpt* nor *kptrlatt* are defined (see explanations with the input variable *prtkpt*). It will allow to select a zero–, one–, two– or three–dimensional grid of k–points. The coordinate of the k–points along vacuum directions is automatically set to zero.

If *vacuum* is not defined, the input variable *vacwidth* will be used to determine automatically whether the distance between atoms is sufficient to have the presence or absence of *vacuum*.

7.5.55 vacwidth

Mnemonics: VACuum WIDTH
Characteristic: LENGTH
Variable type: real parameter
Default value is 10.0

Give a minimum "projected" distance between atoms to be found in order to declare that there is some vacuum present for each of the three directions. By default, given in bohr atomic units (1 Bohr=0.5291772083 Å), although Angstrom can be specified, if preferred, since *vacwidth* has the 'LENGTH' characteristics.

The precise requirement is that a slab of width *vacwidth*, delimited by two planes of constant reduced coordinates in the investigated direction, must be empty of atoms.

7.6 GW variables, VARGW

7.6.1 bdgw

Mnemonics: BanDs for GW calculation
Characteristic: GW
Variable type: integer bdgw(2,*nkptgw*)
Default is all 0's

For each k–point with number igwpt in the range (1:*nkptgw*), *bdgw*(1,igwpt) is the number of the lowest band for which the GW computation must be done, and *bdgw*(2,igwpt) is the number of the highest band for which the GW computation must be done.

7.6.2 ecuteps

Mnemonics: Energy CUT–off for EPSilon (the dielectric matrix)
Characteristic: GW
Variable type: real
Default: 0.0

Only relevant if *optdriver*=3, that is, GW calculations.

ecuteps determines the cut–off energy of the planewave set used to represent the independent–particle susceptibility $\chi^{(0)}_{KS}$, the dielectric matrix ϵ, and its inverse. It is not worth to take *ecuteps* bigger than four times *ecutwfn*, this latter limit corresponding to the highest Fourier components of a wavefunction convoluted with itself. Usually, even twice the value of *ecutwfn* might overkill. In any case, a convergence study is worth.

This set of planewaves can also be determined by the other input variables *npweps* and *nsheps*, but these are much less convenient to use for general systems, than the selection criterion based on a cut–off energy.

7.6.3 ecutsigx

Mnemonics: Energy CUT–off for MAT??
Characteristic: GW
Variable type: real
Default: 0.0

Only relevant if *optdriver*=4, that is, GW calculations. This input variable was named "ecutmat" prior to v4.3.

ecutsigx determines the cut–off energy of the planewave set used to generate the exchange part of the self–energy operator.

This set of planewaves can also be determined by the other input variables *npwsigx* and *nshsigx*, but these are much less convenient to use for general systems, than the selection criterion based on the cut–off energy.

7.6.4 ecutwfn

Mnemonics: Energy CUT–off for WaveFunctions
Characteristic: GW
Variable type: real
Default: 0.0

Only relevant if *optdriver*=3 or 4, that is, GW calculations.

ecutwfn determines the cut–off energy of the planewave set used to represent the wavefunctions in the formula that generates the independent–particle susceptibility $\chi^{(0)}_{KS}$ (for *optdriver*=3), or the self–energy (for *optdriver*=4).

Usually, *ecutwfn* is smaller than *ecut*, so that the wavefunctions are filtered, and some components are ignored. As a side effect, the wavefunctions are no more normalized, and also, no more orthogonal. Also, the set of plane waves can be much smaller for *optdriver*=3, than for *optdriver*=4, although a convergence study is needed to choose correctly both values.

This set of planewaves can also be determined by the other input variables *npwwfn* and *nshwfn*, but these are much less convenient to use for general systems, than the selection criterion based on the cut–off energy.

7.6.5 gwcalctyp

Mnemonics: GW CALCulation TYPe
Characteristic: GW
Variable type: integer
Default: 0

Only relevant if *optdriver*=3 or 4, that is, GW calculations.
gwcalctyp governs the choice between plasmon–pole approximation or full integration, or ... (to be described), for development purposes.

7.6.6 kptgw

Mnemonics: K–PoinTs for GW calculations
Characteristic: GW
Variable type: real kptgw(3,*nkptgw*)
Default: all 0.0's

For each k–point with number igwpt in the range (1:*nkptgw*), *kptgw*(1,igwpt) is the reduced coordinate of the k–point.
At present, not all k–points are possible. Only those corresponding to the k–point grid defined with the same repetition parameters (*kptrlatt*, or *ngkpt*) than the GS one, but WITHOUT any shift, are allowed.

7.6.7 nbandkss

Mnemonics: Number of BaNDs STOred
Characteristic:
Variable type: integer parameter
Default is 0

This input variable (also called "nbndsto" prior to v4.3) is used for the preparation of a GW calculation: it will be used in a GS run (where *optdriver*=0) to generate a _KSS file. In this run, *nbandkss* should be non–zero. Then, this GS run should be followed with a run where *optdriver*=3.

- If *nbandkss*=0, no _KSS file is created
- If *nbandkss*=−1, all the available eigenstates (energies and eigenfunctions) are stored in a abo_KSS file at the end of the ground state calculation. The number of states is forced to be
- the same for all k–points: it will be the minimum of the number of plane waves over all k–points.
- If *nbandkss* is greater than 0, abinit stores (about) *nbandkss* eigenstates in a abo_KSS file. This number of states is forced to be the same for all k–points.

See *npwkss* for the selection of the number of the planewave components of the eigenstates to be stored.
Very important: for the time being, *istwfk* must be 1 for all the k–points.
For more details about the format of the abo_KSS file, see the routine outkss.f.

7.6.8 npwkss

Mnemonics: Number of planewave COMponents STOred
Characteristic:

Variable type: integer parameter
Default is 0

This input variable (was called "ncomsto" prior to v4.3) is used for the preparation of a GW calculation: the GS run (where *optdriver*=1 and *nbandkss* $\neq$ 0) should be followed with a run where *optdriver*=3. Also, if *nbandkss*=0, no use of *npwkss*.

npwkss defines the number of planewave components of the Kohn–Sham states to build the Hamiltonian, in the routine outkss.f, and so, the size of the matrix, the size of eigenvectors, and the number of available states, to be stored in the abo_KSS file. If it is set to 0, then, the planewave basis set defined by the usual Ground State input variable *ecut* is used to generate the superset of all planewaves used for all k–points. Note that this (large) planewave basis is the same for all k–points.

Very important: for the time being, *istwfk* must be 1 for all the k–points.

7.6.9 nkptgw

Mnemonics: Number of GW PoinTs
Characteristic: GW
Variable type: integer
Default: 0

Only relevant if *optdriver*=4, that is, GW calculations. This input variable was called "ngwpt" in versions before v4.3.

nkptgw gives the number of k–points for which the GW calculation must be done. It is used to dimension *kptgw*.

7.6.10 nomegasrd

Mnemonics: Number of OMEGA to evaluate the Sigma Real axis Derivative
Characteristic: GW
Variable type: integer
Default: 9

Only relevant if *optdriver*=4, that is, GW calculations.

The number of frequencies omega where sigma is calculation around the KS energy on the real axis. From these values, the derivative of Sigma with respect to omega and calculated at the KS energy is evaluated.

7.6.11 npweps

Mnemonics: Number of PlaneWaves for EPSilon (the dielectric matrix)
Characteristic: GW
Variable type: integer
Default: 0

Only relevant if *optdriver*=3, that is, GW calculations.

npweps determines the size of the planewave set used to represent the independent–particle susceptibility $\chi^{(0)}_{KS}$, the dielectric matrix ϵ and its inverse.

See *ecuteps* (preferred over *npweps*) for more information.

7.6.12 npwsigx

Mnemonics: Number of PlaneWaves for SIGma eXchange
Characteristic: GW

Variable type: integer
Default: 0

Only relevant if *optdriver*=4, that is, GW calculations. This input variable wxas previously called "npwmat".

npwsigx determines the cut–off energy of the planewave set used to generate the exchange part of the self–energy operator.

See *ecutsigx* (preferred over *npwsigx*) for more information.

7.6.13 npwwfn

Mnemonics: Number of PlaneWaves for WaveFunctioNs
Characteristic: GW
Variable type: integer
Default: 0

Only relevant if *optdriver*=3 or 4, that is, GW calculations.

npwwfn determines the size of the planewave set used to represent the wavefunctions in the formula that generates the independent–particle susceptibility $\chi^{(0)}_{KS}$.

See *ecutwfn* (preferred over *nshwfn*) for more information.

7.6.14 nsheps

Mnemonics: Number of SHells for EPSilon (the dielectric matrix)
Characteristic: GW
Variable type: integer
Default: 0

Only relevant if *optdriver*=3, that is, GW calculations.

nsheps determines the size of the planewave set used to represent the independent–particle susceptibility $\chi^{(0)}_{KS}$, the dielectric matrix ϵ and its inverse.

See *ecuteps* (preferred over *nsheps*) for more information.

7.6.15 nshsigx

Mnemonics: Number of SHells for MAT??
Characteristic: GW
Variable type: integer
Default: 0

Only relevant if *optdriver*=4, that is, GW calculations.This input variable was named "nshma" prior to v4.3.

nshsigx determines the cut–off energy of the planewave set used to generate the exchange part of the self–energy operator.

See *ecutsigx* (preferred over *nshsigx*) for more information.

7.6.16 nshwfn

Mnemonics: Number of SHells for WaveFunctioNs
Characteristic: GW
Variable type: integer
Default: 0

Only relevant if *optdriver*=3 or 4, that is, GW calculations.

nshwfn determines the number of shells of the planewave set used to represent the wavefunctions in the formula that generates the independent–particle susceptibility $\chi^{(0)}_{KS}$.

See *ecutwfn* (preferred over *nshwfn*) for more information.

7.6.17 omegasrdmax

Mnemonics: OMEGA to evaluate the Sigma Real axis Derivative: MAXimal value
Characteristic: GW
Variable type: real
Default: 1.0 eV

Only relevant if GW calculations.

The maximum distance from the KS energy where to evaluate Sigma. Sigma is evaluated at [KS_energy — maxomegasrd, KS_energy + maxomegasrd] sampled *nomegasrd* times.

7.6.18 ppmfrq

Mnemonics: Plasmon Pole Model FReQuency
Characteristic: ENERGY, GW
Variable type: real
Default: 0.0 Ha

(This input variable was named *plasfrq* prior to v4.3). Only relevant if *optdriver*=3, that is, GW calculations.

The present GW implementation is based on a plasmon–pole model. In this plasmon–pole model, the screening must be available at zero frequency, as well as at another frequency, imaginary, on the order of the plasmon frequency (the peak in the EELS spectrum). This information is used to derive the behaviour of the dielectric matrix for all the frequencies (complex). *ppmfrq* defines the imaginary frequency at which the dielectric matrix is evaluated, in addition to the zero frequency. If the plasmon–pole approximation is good, then, the choice of *ppmfrq* should have no influence on the final result. One should check whether this is the case. In general, the plasmon frequencies of bulk solids are on the order of 0.5 Hartree.

7.6.19 soenergy

Mnemonics: Scissor Operator ENERGY
Characteristic: GW, ENERGY
Variable type: real
Default: 0.0

Only relevant if *optdriver*=3, that is, GW calculations of screening. The Scissor operator energy to be added to unoccupied levels for the screening calculation. In some cases, it mimics a second iteration self–consistent GW calculation.

7.6.20 zcut

Mnemonics: Z–CUT
Characteristic: GW, ENERGY
Variable type: real
Default: 0.1 eV = $3.67493260 \times 10^{-3}$ Ha

Only relevant if *optdriver*=4, that is, GW calculations. It is meant to avoid some divergencies that might occur due to the numerical treatment of integrable poles along the integration path. If

the denominator becomes smaller than *zcut*, a small imaginary part (depending on *zcut*) is added, in order to avoid the divergence.

Ideally, one should make a convergence study of *zcut* decreasing for increasing number of k–points.

7.7 Internal variables, VARINT

7.7.1 kptns

Mnemonics: K–PoinTs re–Normalized and Shifted
Characteristic: INTERNAL
Variable type: real array kptns(3,*nkpt*)

If *nqpt*=0, or if one is doing a reponse calculation, this internal variable is derived from *kpt* and *kptnrm*: *kptns*(1:3,:)= *kpt*(1:3,:)/ *kptnrm*, so that it is *kpt* renormalized by *kptnrm*.

If *nqpt*=1 and one is not doing a ground–state calculation, this internal variable is derived from *kpt*, *kptnrm* and *qptn* *kptns*(1:3,:)= *kpt*(1:3,:)/ *kptnrm*+ *qptn*(1:3), so that it is kpt renormalized by *kptnrm*, then shifted by *qptn*(1:3).

7.7.2 mband

Mnemonics: Maximum number of BANDs
Characteristic: INTERNAL
Variable type: integer

This internal variable deduces from *nband*(1:*nkpt***nsppol*) the maximum number of bands, over all k–points and spin–polarisation.

7.7.3 mgfft

Mnemonics: Maximum of nGFFT
Characteristic: INTERNAL
Variable type: integer

This internal variable gives the maximum of *ngfft*(1:3).

7.7.4 mpw

Mnemonics: Maximum number of Plane Waves
Characteristic: INTERNAL
Variable type: integer

This internal variable gives the maximum of the number of plane waves over all k–points. It is computed from *ecut*, and the description of the cell, provided by *acell*, *rprim*, and/or *angdeg*.

7.7.5 nelect

Mnemonics: Number of ELECTrons
Characteristic: INTERNAL
Variable type: real number

This internal variable gives the number of electrons per unit cell, as computed from the sum of the valence electrons related to each atom (given by the pseudopotential), that is called “zval” and the input variable *charge*: *nelect*=zval–charge.

7.7.6 nfft

Mnemonics: Number of FFT points
Characteristic: INTERNAL
Variable type: integer

If the space parallelisation is not used, this internal variable gives the number of Fast Fourier Transform points, in the grid generated by *ngfft*(1:3). It is simply the product of the three components of *ngfft*.

If the space parallelisation is used, then it becomes the number of Fast Fourier Transform points attributed to this particular processor. It is no more the above–mentioned simple product, but a number usually close to this product divided by the number of processors on which the space is shared.

7.7.7 qptn

Mnemonics: Q–PoinT re–Normalized
Characteristic: INTERNAL
Variable type: real array qptn(3)

Used if *nqpt*=1. This internal variable is derived from *qpt* and *qptnrm*: *qptn*(1:3)= *qpt*(1:3)/ *qptnrm*.

7.7.8 usepaw

Mnemonics: USE Projector Augmented Waves method
Characteristic: INTERNAL
Variable type: integer parameter
Value is set by the pseudopotential files: either PAW or norm–conserving.

If the user wants to use the Projector Augmented Method, then *usepaw* must be 1. This variable is determined by the pseudopotentials files. Indeed, special PAW pseudo–projector files must be used (these are the equivalent of the pseudopotential files for the pseudopotential method). Such files are NOT yet available for the entire periodic table. Also, forces and stresses are not yet computed in the PAW method as implemented in ABINITv4.1. Other limitations are present as well, precluding the use of PAW for real production work. These limitations will be waived as time passes.

Related variables: *pawecutdg*, *pawlcutd*, *pawmqgrdg*, *pawnphi*, *pawntheta*.

7.8 Parallelisation variables, VARPAR

7.8.1 localrdwf

Mnemonics: LOCAL ReaD WaveFunctions
Characteristic: DEVELOP, PARALLEL
Variable type: integer
Default is 1.

This input variable is used only in abinip. If *localrdwf*=1, the input wavefunction disk file is read locally by each processor, while if *localrdwf*=0, only one processor reads it, and BCAST it to other processors.

The option *localrdwf*=0 is NOT allowed when *mkmem*==0 (or, for RF, when *mkqmem*==0, or *mk1mem*==0), that is, when the wavefunctions are stored on disk. This is still to be coded ...

In the case of a parallel computer with a unique file system, both options are as convenient for the user. However, if the I/O are slow compared to communications between processors, (e.g. for CRAY T3E machines), *localrdwf*=0 should be much more efficient; if you really need temporary disk storage, switch to *localrdwf*=1).

In the case of a cluster of nodes, with a different file system for each machine, the input wavefunction file must be available on all nodes if *localrdwf*=1, while it is needed only for the master node if *localrdwf*=0.

7.9 Projector–Augmented Wave variables, VARPAW

7.9.1 ngfftdg

Mnemonics: Number of Grid points for Fast Fourier Transform: Double Grid
Characteristic:
Variable type: integer array ngfftdg(3)
Default:

Needed only when *usepaw*=1.
To be documented.

7.9.2 pawecutdg

Mnemonics: PAW - Energy CUToff for the Double Grid
Characteristic: ENERGY
Variable type: real parameter
Default: 1.25 times ecut

Needed only when *usepaw*=1.

Define the energy cut–off for the fine FFT grid (that allow to transfer data from the normal, coarse, FFT grid to the spherical grid around each atom).

pawecutdg must be larger or equal to *ecut*. If equal to it, then no fine grid is used. The results are not very accurate, but the computations proceed quite fast.

The default value is sometimes a bit too low, but does not slow down the computation. The choice made for this variable DOES have a bearing on the numerical accuracy of the results, and, as such, should be the object of a convergence study. The convergence test might be made on the total energy or derived quantities, like forces, but also on the two values of the "Compensation charge inside spheres", a quantity written in the log file.

7.9.3 pawlcutd

Mnemonics: PAW - 1+L angular momentum used to CUT the development in moments of the Densitites
Characteristic:
Variable type: integer parameter
The default is 10

Needed only when *usepaw*=1.

The expansion of the densities in angular momenta is performed up to l=*pawlcutd*-1.

The choice made for this variable DOES have a bearing on the numerical accuracy of the results, and, as such, should be the object of a convergence study. The convergence test might be made on the total energy or derived quantities, like forces, but also on the two values of the "Compensation charge inside spheres", a quantity written in the log file.

7.9.4 pawmqgrdg

Mnemonics: PAW - Max. number of Q–space GRid points for psp for the Double Grid
Characteristic:
Variable type: integer parameter
The default is -1.

Needed only when *usepaw*=1.
Same use as *mqgrid*, but for the fine grid, instead of the coarse grid.
If set to -1, the step corresponding to *mqgrid* for the coarse grid (defined by the energy cut–off *ecut*) is computed, and then, the same step is used for the fine grid (defined by the energy cut–off *pawecutdg*).

7.9.5 pawnphi

Mnemonics: PAW - Number of PHI angles used to discretize the sphere around each atom.
Characteristic:
Variable type: integer parameter
The default is 13.

Needed only when *usepaw*=1.
Number of phi angles (longitude) used to discretize the data on the atomic spheres. This discretization is completely defined by *pawnphi* and *pawntheta*.

7.9.6 pawntheta

Mnemonics: PAW - Number of THETA angles used to discretize the sphere around each atom.
Characteristic:
Variable type: integer parameter
The default is 12

Needed only when *usepaw*=1.
Number of theta angles (latitude) used to discretize the data on the atomic spheres. This discretization is completely defined by *pawntheta* and *pawnphi*.

7.10 Response Function variables, VARRF

7.10.1 dsifkpt

Mnemonics: DenSiFy K–PoinTs
Characteristic:
Variable type: integer array dsifkpt(3)
Default is 1.

Can be used to density the k–point grid along the lines that are parallel to the three primitive vectors, in reciprocal space. Should be useful for third–order derivatives that include some derivative with respect to k–points or electric field. This part is in development. For the time being, consult ABINIT/Infos/nonlinear.ps

7.10.2 mkqmem

Mnemonics: Maximum number of K+Q - points in MEMory

7.10.3 mk1mem

Mnemonics: Maximum number of K - points for 1st order wavefunctions, kept in MEMory
Characteristic: RESPFN
Variable type: integer parameters
Default is *nkpt*, i.e. in–core solution.

Plays a role similar to *mkmem* but for different sets of wavefunctions: the ground state wavefunctions at k+q and the first–order wavefunctions. Only needed for response calculations. Internal representation as *mkmem*s(2) and *mkmem*s(3).

Note (991019) that although the effective number of k–points can be reduced thanks to symmetry for different perturbations, *mkqmem* and *mk1mem* are presently still compared with the input *nkpt*.

This should be changed later.

7.10.4 prepanl

Mnemonics: PREPAre Non–Linear response calculation
Characteristic: RESPFN
Variable type: integer parameter
Default is 0.

The computation of third–order derivatives from the 2n+1 theorem requires the first–order wavefunctions and densities obtained from a linear response calculation. The standard approach in a linear response calculation is (i) to compute only the irreductible perturbations, and (ii) to use symmetries to reduce the number of k–points for the k–point integration.

This approach cannot be applied, presently (v4.1), if the first–order wavefunctions are to be used to compute third–order derivatives. First, for electric fields, the code needs the derivatives along the three directions. Still, in case of phonons, only the irreducible perturbations are required. Second, for both electric fields and phonons, the wavefunctions must be available in half the BZ (*kptopt*=2), or the full BZ (*kptopt*=3).

During the linear response calculation, in order to prepare a non–linear calculation, one should put *prepanl* to 1 in order to force ABINIT (i) to compute the electric field perturbation along the three directions explicitely, and (ii) to keep the full number of k–points.

7.10.5 prtbbb

Mnemonics: PRinT Band–By–Band decomposition
Characteristic: RESPFN
Variable type: integer parameter
Default is 0.

If *prtbbb* is 1, print the band–by–band decomposition of Born effective charges and localization tensor, in case they are computed. See Ph. Ghosez and X. Gonze, J. Phys.: Condens. Matter 12, 9179 (2000), and M. Veithen, X. Gonze and Ph. Ghosez, to be published.

7.10.6 rfasr

Mnemonics: Response Function: Acoustic Sum Rule
Characteristic: RESPFN
Variable type: integer parameter
Default is 0.

Control the evaluation of the acoustic sum rule in effective charge calculations within a response function calculation.

- 0 ⇒ no acoustic sum rule imposed
- 1 ⇒ acoustic sum rule imposed with extra charge evenly distributed among atoms
- 2 ⇒ acoustic sum rule imposed with extra charge given proportionally to those atoms with the largest effective charge.

7.10.7 rfatpol

Mnemonics: Response Function: limits of ATomic POLarisations
Characteristic: RESPFN

7.10.8 rf1atpol

Mnemonics: non–linear Response Function, 1st mixed perturbation: limits of ATomic POLarisations
Characteristic: NON–LINEAR

7.10.9 rf2atpol

Mnemonics: non–linear Response Function, 2nd mixed perturbation: limits of ATomic POLarisations
Characteristic: NON–LINEAR

7.10.10 rf3atpol

Mnemonics: non–linear Response Function, 3rd mixed perturbation: limits of ATomic POLarisations
Characteristic: NON–LINEAR

Variable type: integer array of 2 elements
Default is 1 1

Control the range of atoms for which displacements will be considered in phonon calculations (atomic polarisations), or in non–linear computations, using the 2n+1 theorem.

These values are only relevant to phonon response function calculations, or non–linear computations.

May take values from 1 to *natom*, with *rfatpol*(1) ≤ *rfatpol*(2).

The atoms to be moved will be defined by the

do–loop variable iatpol:

do iatpol=*rfatpol*(1),*rfatpol*(2)

For the calculation of a full dynamical matrix, use *rfatpol*(1)=1 and *rfatpol*(2)=*natom*, together with *rfdir* 1 1 1. For selected elements of the dynamical matrix, use different values of *rfatpol* and/or *rfdir*. The name 'iatpol' is used for the part of the internal variable ipert when it runs from 1 to *natom*. The internal variable ipert can also assume values larger than *natom*, of electric field or stress type (see respfn.help).

7.10.11 rfdir

Mnemonics: Response Function: DIRections
Characteristic: RESPFN

7.10.12 rf1dir

Mnemonics: non–linear Response Function, 1st mixed perturbation: DIRections
Characteristic: NON–LINEAR

7.10.13 rf2dir

Mnemonics: non–linear Response Function, 2nd mixed perturbation: DIRections
Characteristic: NON–LINEAR

7.10.14 rf3dir

Mnemonics: non–linear Response Function, 3rd mixed perturbation: DIRections
Characteristic: NON–LINEAR

Variable type: integer array of 3 elements
Default is 0 0 0.

Gives the directions to be considered for response function calculations, or non–linear computations.

The three elements corresponds to the three primitive vectors, either in real space (phonon calculations), or in reciprocal space (d/dk and homogeneous electric field calculations). So, they generate a basis for the generation of the dynamical matrix or to macroscopic didlectric tensor, of the effective charge tensors.

If equal to 1, response functions, as defined by *rfelfd*, *rfphon*, *rfdir* and *rfatpol*, are to be computed for the corresponding direction. If 0, this direction should not be considered (for non–linear computations, the corresponding input variables should be used).

7.10.15 rfelfd

Mnemonics: Response Function with respect to the ELectric FielD
Characteristic: RESPFN

7.10.16 rf1elfd

Mnemonics: non–linear Response Function, 1st mixed perturbation: ELectric FielD
Characteristic: NON–LINEAR

7.10.17 rf2elfd

Mnemonics: non–linear Response Function, 2nd mixed perturbation: ELectric FielD
Characteristic: NON–LINEAR

7.10.18 rf3elfd

Mnemonics: non–linear Response Function, 3rd mixed perturbation: ELectric FielD
Characteristic: NON–LINEAR

Variable type: integer parameter
Default is 0.

Turns on electric field response function calculations (or non–linear computation, including the electric field perturbation). Actually, such calculations requires first the non–self–consistent calculation of derivatives with respect to k, independently of the electric field perturbation itself.

- 0 ⇒ no electric field perturbation
- 1 ⇒ full calculation, with first the derivative of ground–state wavefunction with respect to k (d/dk calculation), by a non–self–consistent calculation, then the generation of the
- first–order response to an homogeneous electric field
- 2 ⇒ only the derivative of ground–state wavefunctions with respect to k;
- 3 ⇒ only the generation of the first–order response to the electric field, assuming that the data on derivative of ground–state wavefunction with respect to k is available on disk.

(Note: because the tolerances to be used for derivatives or homogeneous electric field are different, one often does the calculation of derivatives in a separate dataset, followed by calculation of electric field response as well as phonon. The options 2 and 3 proves useful in that context; also, in case a scissor shift is to be used, it is usually not applied for the d/dk response).

7.10.19 rfmeth

Mnemonics: Response Function METHod
Characteristic: RESPFN
Variable type: integer parameter
Default is 1.

Selects method used in response function calculations. Presently, only 1 is allowed.

7.10.20 rfphon

Mnemonics: Response Function with respect to PHONons
Characteristic: RESPFN

7.10.21 rf1phon

Mnemonics: non–linear Response Function, 1st mixed perturbation: PHONons
Characteristic: NON–LINEAR

7.10.22 rf2phon

Mnemonics: non–linear Response Function, 2nd mixed perturbation: PHONons
Characteristic: NON–LINEAR

7.10.23 rf2phon

Mnemonics: non–linear Response Function, 3rd mixed perturbation: PHONons
Characteristic: NON–LINEAR

Variable type: integer parameter
Default is 0.

It must be equal to 1 to run phonon response function calculations, or to include some phonon perturbation in non–linear computations.

7.10.24 rfstrs

Mnemonics: Response Function with respect to STRainS
Characteristic: RESPFN
Variable type: integer parameter
Default is 0.

Used to run strain response–function calculations (e.g. needed to get elastic constants). Define, with *rfdir*, the set of perturbations.

- 0 ⇒ no strain perturbation
- 1 ⇒ only uniaxial strain(s) (ipert=*natom*+3 is activated)
- 2 ⇒ only shear strain(s) (ipert=*natom*+4 is activated)
- 3 ⇒ both uniaxial and shear strain(s) (both ipert=*natom*+3 and ipert=*natom*+4 are activated)

See the possible restrictions on the use of strain perturbations, in the `respfn_help` file.

7.10.25 rfthrd

Mnemonics: Response Function of THiRD order
Characteristic: RESPFN
Variable type: integer parameter
Default is 0.

Used to control response function calculation of third order response. Not implemented.

7.10.26 rfuser

Mnemonics: Response Function, USER–defined
Characteristic: RESPFN
Variable type: integer parameter
Default is 0.

Available to the developpers, to activate the use of ipert=*natom*+5 and ipert=*natom*+6, two sets of perturbations that the developpers can define.

- 0 ⇒ no computations for ipert=*natom*+5 or ipert=*natom*+6
- 1 ⇒ response with respect to perturbation *natom*+5 will be computed
- 2 ⇒ response with respect to perturbation *natom*+6 will be computed
- 3 ⇒ responses with respect to perturbations *natom*+5 and *natom*+6 will be computed

In order to define and use correctly the new perturbations, the developper might have to include code lines or additional routines at the level of the following routines: cgwf3.f, chkph3.f, dyout3.f, d2sym3.f, eneou3.f, eneres3.f, gath3.f, insy3.f, loper3.f, mkcor3.f, nstdy3.f, nstwf3.f, respfn.f, scfcv3.f, syper3.f, vloca3.f, vtorho3.f, vtowfk3.f, wings3.f,. In these routines, the developper should pay a particular attention to the rfpert array, defined in the routine respfn.f, as well as to the ipert local variable.

7.10.27 sciss

Mnemonics: SCISSor operator
Characteristic: RESPFN, ENERGY
Variable type: real parameter
Default is 0.

It is the value of the "scissors operator", the shift of conduction band eigenvalues, used in response function calculations. Can be specified in Ha (the default), Ry, eV or Kelvin, since *ecut* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV)

Typical use is for response to electric field (*rfelfd*=3), but NOT for d/dk (*rfelfd*=2) and phonon responses.

7.10.28 td_maxene

Mnemonics: Time–Dependent dft: MAXimal kohn–sham ENErgy difference
Characteristic: TDDFT
Variable type: real parameter
Default is huge.

The Matrix to be diagonalized in the Casida framework (see "Time–Dependent Density Functional Response Theory of Molecular systems: Theory, Computational Methods, and Functionals", by M.E. Casida, in Recent Developments and Applications of Modern Density Functional Theory, edited by J.M. Seminario (Elsevier, Amsterdam, 1996).) is a NxN matrix, where, by default, N is the product of the number of occupied states by the number of unoccupied states.

The input variable *td_maxene* allows to diminish N: it selects only the pairs of occupied and unoccupied states for which the Kohn–Sham energy difference is less than *td_maxene*.

See *td_mexcit* for an alternative way to decrease N.

7.10.29 td_mexcit

Mnemonics: Time–Dependent dft: Maximal number of EXCITations
Characteristic: TDDFT
Variable type: real parameter
Default is 0.

The Matrix to be diagonalized in the Casida framework (see "Time–Dependent Density Functional Response Theory of Molecular systems: Theory, Computational Methods, and Functionals", by M.E. Casida, in Recent Developments and Applications of Modern Density Functional Theory, edited by J.M. Seminario (Elsevier, Amsterdam, 1996).) is a NxN matrix, where, by default, N is the product of the number of occupied states by the number of unoccupied states.

The input variable *td_mexcit* allows to diminish N: it selects the first *td_mexcit* pairs of occupied and unoccupied states, ordered with respect to increasing Kohn–Sham energy difference. However, when *td_mexcit* is zero, all pairs are allowed.

See *td_maxene* for an alternative way to decrease N.

7.11 Structure optimization variables, VARRLX

7.11.1 amu

Mnemonics: Atomic Mass Units
Characteristic:
Variable type: real array amu(*ntypat*)

Default is provided by a database of atomic masses.

Gives the masses in atomic mass units for each kind of atom in cell. These masses are used in performing molecular dynamical atomic motion if *ionmov*=1, 6, 7 or 8. Note that one may set all masses to 1 for certain cases in which merely structural relaxation is desired and not actual molecular dynamics.

Using 1986 recommended values, 1 atomic mass unit = 1.6605402e-27 kg. In this unit the mass of Carbon 12 is exactly 12.

A database of atomic masses is provided, giving default values. Note that the default database uses mixed isotope masses (for Carbon the natural occurence of Carbon 13 is taken into account). The values are those recommended by the commission on Atomic Weights and Isotopic Abundances, Inorganic Chemistry Division, IUPAC, in Pure Appl. Chem. 60, 841 (1988). For Tc, Pm, Po to Ac, Pa and beyond U, none of the isotopes has a half–life greater than 3.0×10^{10} years, and the values provided in the database do not come from that source. For alchemical pseudoatoms, the masses of the constituents atoms are mixed, according to the alchemical miwing coefficients *mixalch*

7.11.2 delayperm

Mnemonics: DELAY between trials to PERMUTE atoms
Characteristic:
Variable type: integer
Default is 0.

Delay (number of time steps) between trials to permute two atoms, in view of accelerated search of minima. Still in development. See the routine moldyn.f. See also *signperm*. When *delayperm* is zero, there is not permutation trials.

7.11.3 dilatmx

Mnemonics: DILATation: MaXimal value
Characteristic:
Variable type: real parameter
Default is 1.0.

Gives the maximal permitted scaling of the lattice parameters when the cell shape and dimension is varied (see variable *optcell*). It is used to define the sphere of plane waves and FFT box coherent with the possible modifications of the cell (*ionmov*==2 and *optcell* $\neq$ 0). For these definitions, it is equivalent to changing *ecut* by multiplying it by $dilatmx^2$ (the result is an "effective *ecut*", called internally "ecut_eff", other uses of *ecut* being not modified when *dilatmx*>1.0.

Using *dilatmx*¡1.0 is equivalent to changing *ecut* in all its uses. This is allowed, although its meaning is no longer related to a maximal expected scaling.

Setting *dilatmx* to a large value leads to waste of CPU time and memory. Supposing you think that the optimized *acell* values might be 10% larger than your input values, use simply *dilatmx* 1.1.

This will already lead to an increase of the number of planewaves by a factor (1.1)3=1.331, and a corresponding increase in CPU time and memory. It is possible to use *dilatmx* when *optcell*=0, but a value larger than 1.0 will be a waste.

Must be 1.0 for RF calculations.

7.11.4 dtion

Mnemonics: Delta Time for IONs
Characteristic:

Variable type: real parameter
Default is 100.

Used for controlling ion time steps. If *ionmov* is set to 1, 6 or 7, then molecular dynamics is used to update atomic positions in response to forces. The parameter *dtion* is a time step in atomic units of time. (One atomic time unit is 2.418884e-17 seconds, which is the value of Planck's constant in hartree*sec.) In this case the atomic masses, in amu (given in array "amu"), are used in Newton's equation and the viscosity (for *ionmov*=1) and number of time steps are provided to the code using input variables "*vis*" and "*ntime*". The code actually converts from masses in amu to masses in atomic units (in units of electron masses) but the user enters masses in amu. (The conversion from amu to atomic units (electron masses) is 1822.88851 electron masses/amu.) A typical good value for *dtion* is about 100. The user must try several values for *dtion* in order to establish the stable and efficient choice for the accompanying amu, atom types and positions, and *vis* (viscosity).

For quenched dynamics (*ionmov*=7), a larger time step might be taken, for example 200.

No meaning for RF calculations.

7.11.5 ecutsm

Mnemonics: Energy CUToff SMearing
Characteristic: ENERGY
Variable type: real parameter (in Hartree)
Default is 0.d0

This input variable is important when performing relaxation of unit cell size and shape (non–zero *optcell*). Using a non–zero *ecutsm*, the total energy curves as a function of *ecut*, or *acell*, can be smoothed, keeping consistency with the stress (and automatically including the Pulay stress). The recommended value is 0.5 Ha. Actually, when *optcell* $\neq 0$, ABINIT requires *ecutsm* to be larger than zero. If you want to optimize cell shape and size without smoothing the total energy curve (a dangerous thing to do), use a very smalle *ecutsm*, on the order of one microHartree.

Technical information: See Bernasconi et al, J. Phys. Chem. Solids 56, 501 (1995) for a related method.

ecutsm allows to define an effective kinetic energy for plane waves, close to, but lower than the maximal kinetic energy *ecut*. For kinetic energies less than *ecut*-*ecutsm*, nothing is modified, while between *ecut*-*ecutsm* and *ecut*, the kinetic energy is multiplied by: $1.0/(x^2(3-2*x))$ where x = (*ecut* - kinetic_energy)/*ecutsm*

Note that x2 (3-2*x) is 0 at x=0, with vanishing derivative, and that at x=1, it is 1, with also vanishing derivative.

If *ecutsm* is zero, the unmodified kinetic energy is used.

ecutsm can be specified in Ha (the default), Ry, eV or Kelvin, since *ecutsm* has the 'ENERGY' characteristics. (1 Ha=27.2113961 eV).

A few test for Silicon (diamond structure, 2 k–points) have shown 0.5 Ha to be largely enough for *ecut* between 2Ha and 6Ha, to get smooth curves. It is likely that this value is OK as soon as *ecut* is larger than 4Ha.

7.11.6 friction

Mnemonics: internal FRICTION coefficient
Characteristic:
Variable type: real parameter
Default is 0.001

Gives the internal *friction* coefficient (atomic units) for Langevin dynamics (when *ionmov*=9): fixed temperature simulations with random forces.

The equation of motion is:

$$M_I d^2 R_I/dt^2 = F_I - \text{friction} M_I dR_I/dt - F_{\text{random } I}$$

where $F_{\text{random } I}$ is a Gaussian random force with average zero, and variance 2 friction$M_I kT$.

The atomic unit of *friction* is Hartrees*electronic mass*(atomic time units)/$bohr^2$. See J. Chelikowsky, J. Phys. D: Appl Phys. 33 (2000) R33.

7.11.7 getcell

Mnemonics: GET CELL parameters from ...
Characteristic:
Variable type: integer parameter, an instance of a 'get' variable.
Default is 0.

This variable is typically used to chain the calculations, in the multi–dataset mode (*ndtset*>0), since it describes from which dataset *acell* and *rprim* are to be taken, as input of the present dataset. The cell parameters are EVOLVING variables, for which such a chain of calculations is useful.

If ==0, no use of previously computed values must occur.

If it is positive, its value gives the index of the dataset from which the data are to be used as input data. It must be the index of a dataset already computed in the SAME run.

If equal to −1, the output data of the previous dataset must be taken, which is a frequently occuring case. However, if the first dataset is treated, −1 is equivalent to 0, since no dataset has yet been computed in the same run.

If another negative number, it indicates the number of datasets to go backward to find the needed data (once again, going back beyond the first dataset is equivalent to using a null get variable).

7.11.8 getxcart

Mnemonics: GET XCART from ...

7.11.9 getxred

Mnemonics: GET XRED from ...

7.11.10 getvel

Mnemonics: GET VEL from ...
Characteristic:
Variable type: integer parameters, instances of 'get' variables
Default is 0.

These variables are typically used to chain the calculations, in the multi–dataset mode (*ndtset* > 0) since they describe from which dataset the corresponding output variables are to be taken, as input of the present dataset. The atomic positions and velocities are EVOLVING variables, for which such a chain of calculation is useful.

Note that the use of *getxcart* and *getxred* differs when *acell* and *rprim* are different from one dataset to the other.

If ==0, no use of previously computed values must occur.

If it is positive, its value gives the index of the dataset from which the data are to be used as input data. It must be the index of a dataset already computed in the SAME run.

If equal to −1, the output data of the previous dataset must be taken, which is a frequently occuring case. However, if the first dataset is treated, −1 is equivalent to 0, since no dataset has yet been computed in the same run.

If another negative number, it indicates the number of datasets to go backward to find the needed data (once again, going back beyond the first dataset is equivalent to using a null get variable).

Note: *getxred* and *getxcart* cannot be simultaneously non–zero for the same dataset. On the other hand the use of *getvel* with *getxred* is allowed, despite the different coordinate system.

7.11.11 iatcon

Mnemonics: Indices of AToms in CONstraint equations
Characteristic: NO MULTI, NOT INTERNAL
Variable type: integer array iatcon(*natcon*,*nconeq*)
Default is 0

Gives the indices of the atoms appearing in each of the *nconeq* independent equations constraining the motion of atoms during structural optimization or molecular dynamics (see *nconeq*, *natcon*, and *wtatcon*).

(Note: combined with *wtatcon* to give internal representation of the latter — this should be described)

7.11.12 iatfix

Mnemonics: Indices of AToms that are FIXed

7.11.13 iatfixx

Mnemonics: Indices of AToms that are FIXed along the X direction

7.11.14 iatfixy

Mnemonics: Indices of AToms that are FIXed along the Y direction

7.11.15 iatfixz

Mnemonics: Indices of AToms that are FIXed along the Z direction
Characteristic: *iatfixx*,*iatfixy* and *iatfixz* are NOT INTERNAL
Variable type: integer arrays of length *natfix*, *natfixx*, *natfixy* or *natfixz*
No Default (ignored unless *natfix*, *natfixx*, *natfixy* or *natfixz* > 0).

Give the index (in the range 1 to *natom*) of each atom which is to be held fixed for structural optimization or molecular dynamics. The variable *iatfix* lists those fixed in the three directions, while the other variables allow to fix some atoms along x, y or z directions, or a combination of these.

WARNING: The implementation is inconsistent!! For *ionmov*==1, the fixing of directions was done in cartesian coordinates, while for the other values of *ionmov*, it was done in reduced coordinates. Sorry for this.

There is no harm in fixing one atom in the three directions using *iatfix*, then fixing it again in other directions by mentioning it in *iatfixx*, *iatfixy* or *iatfixz*.

The internal representation of these input data is done by the mean of one variable *iatfix*(3,*natom*), defined for each direction and each atom, being 0 if the atom is not fixed along

the direction, and 1 if the atom is fixed along the direction. When some atoms are fixed along 1 or 2 directions, the use of symmetries is restricted to symmetry operations whose (3x3) matrices *symrel* are diagonal.

If the geometry builder is used, *iatfix* will be related to the preprocessed set of atoms, generated by the geometry builder. The user must thus foresee the effect of this geometry builder (see *objarf*).

7.11.16 ionmov

Mnemonics: IONic MOVEs
Characteristic:
Variable type: integer parameter
Default for ionmov is 0.

Control the displacements of ions, and eventually (see *optcell*) changes of cell shape and size.

- 0 ⇒ do not move ions;
- 1 ⇒ move atoms using molecular dynamics with optional viscous damping (friction linearly proportional to velocity). The viscous damping is controlled by the parameter "*vis*". If actual undamped molecular dynamics is desired, set *vis* to 0. The implemented algorithm is the generalisation of the Numerov technique (6th order), but is NOT invariant upon time–reversal, so that the energy is not conserved. The value *ionmov*=6 will usually be preferred, although the algorithm that is implemented is lower–order. opcell $\neq$ 0 is not available
- 2 ⇒ conduct structural optimization using the Broyden–Fletcher–Goldfarb–Shanno minimization (BFGS). This is much more efficient for structural optimization than viscous damping, when there are less than let's say 10 degrees of freedom to optimize.
- 3 ⇒ conduct structural optimization using the Broyden–Fletcher–Goldfarb–Shanno minimization (BFGS), modified to take into account the total energy as well as the gradients (as in usual BFGS). See the paper by Schlegel, J. Comp. Chem. 3, 214 (1982). Might be better than *ionmov*=2 for few degrees of freedom (less than 3 or 4)
- 4 ⇒ conjugate gradient algorithm for simultaneous optimization of potential and ionic degrees of freedom. It can be used with *iscf*=2 and *iscf*=5 or 6 (WARNING: this is under development, and does not work very well in many cases). *optcell* $\neq$ 0 is not available.
- 5 ⇒ Simple relaxation of ionic positions according to (converged) forces. Equivalent to *ionmov*=1 with zero masses, albeit the relaxation coefficient is not *vis*, but *iprcfc*. *optcell* $\neq$ 0 is not available.
- 6 ⇒ Molecular dynamics using the Verlet algorithm, see Allen and Tildesley "Computer simulation of liquids" 1987, p 81. Although partly coded, *optcell* $\neq$ 0 is not available. The only related parameter is the time step (*dtion*).
- 7 ⇒ Quenched Molecular dynamics using the Verlet algorithm, and stopping each atom for which the scalar product of velocity and force is negative. Although partly coded, *optcell* $\neq$ 0 is not available. The only related parameter is the time step (*dtion*). The goal is not to produce a realistic dynamics, but to go as fast as possible to the minimum. For this purpose, it is advised to set all the masses to the same value (for example, use the Carbon mass, i.e. set *amu* to 12 for all type of atoms).
- 8⇒ Molecular dynamics with Nose–Hoover thermostat, using the Verlet algorithm. Although partly coded, *optcell* $\neq$ 0 is not available. Related parameters: the time step (*dtion*), the initial temperature (*mditemp*), the final temperature (*mdftemp*), and the thermostat mass (*noseinert*).

- 9⇒ Langevin molecular dynamics. Although partly coded, *optcell* $\neq 0$ is not available. Related parameters: the time step (*dtion*), the initial temperature (*mditemp*), the final temperature (*mdftemp*), and the friction coefficient (*friction*).

No meaning for RF calculations.

7.11.17 mdftemp

Mnemonics: Molecular Dynamics Final Temperature
Characteristic:
Variable type: real mdftemp
Default is mdftemp=*mditemp*

Give the final temperature (for itime=*ntime*) of the Nose–Hoover thermostat (*ionmov*=8) and Langevin dynamics (*ionmov*=9), in Kelvin. This temperature will change linearly from *mditemp* at itime=1 to the final temperature *mdftemp* at the end of the *ntime* timesteps.

7.11.18 mditemp

Mnemonics: Molecular Dynamics Initial Temperature
Characteristic:
Variable type: real mditemp
Default is 300

Give the initial temperature (for itime=1) of the Nose–Hoover thermostat (*ionmov*=8) and Langevin dynamics (*ionmov*=9), in Kelvin. This temperature will change linearly to reach the temperature *mdftemp* at the end of the *ntime* timesteps.

7.11.19 mdwall

Mnemonics: Molecular Dynamics WALL location
Characteristic:
Variable type: real parameter
Default is 10000.0 Bohr (the walls are extremely far away).

Gives the location (atomic units) of walls on which the atoms will bounce back. when *ionmov*=6, 7, 8 or 9. For each cartesian direction idir=1, 2 or 3, there is a pair of walls with coordinates *xcart*(idir)=-wall and *xcart*(idir)=*rprim*d(idir,idir)+wall. Supposing the particle will cross the wall, its velocity normal to the wall is reversed, so that it bounces back.

By default, given in bohr atomic units (1 bohr=0.5291772083 Å), although Angstrom can be specified, if preferred, since *mdwall* has the 'LENGTH' characteristics.

7.11.20 natcon

Mnemonics: Number of AToms in CONstraint equations
Characteristic: NO MULTI
Variable type: integer array of length *nconeq*
Default is 0

Gives the number of atoms appearing in each of the *nconeq* independent equations constraining the motion of atoms during structural optimization or molecular dynamics (see *nconeq*, *iatcon*, and *wtatcon*).

7.11.21 natfix

Mnemonics: Number of Atoms that are FIXed

7.11.22 natfixx

Mnemonics: Number of Atoms that are FIXed along the X direction

7.11.23 natfixy

Mnemonics: Number of Atoms that are FIXed along the Y direction

7.11.24 natfixz

Mnemonics: Number of Atoms that are FIXed along the Z direction
Characteristic: NOT INTERNAL
Variable type: integer parameter
Defaults are 0 (no atoms held fixed).

Gives the number of atoms (not to exceed *natom*) which are to be held fixed during a structural optimization or molecular dynamics.

When *natfix* > 0, *natfix* entries should be provided in array *iatfix*.

When *natfixx* > 0, *natfixx* entries should be provided in array *iatfixx*, and so on ...

7.11.25 nconeq

Mnemonics: Number of CONstraint EQuations
Characteristic: NO MULTI
Variable type: integer parameter
Default is 0

Gives the number of independent equations constraining the motion of atoms during structural optimization or molecular dynamics (see *natcon*, *iatcon*, and *wtatcon*).

7.11.26 noseinert

Mnemonics: NOSE INERTia factor
Characteristic:
Variable type: real noseinert
Default is 1.0d5

Give the inertia factor WT of the Nose–Hoover thermostat (when *ionmov*=8), in atomic units of weight $*$ length^2, that is (electron mass) $*$ (bohr)^2. The equations of motion are:

$$M_I \frac{d^2 R_I}{dt^2} = F_I - \frac{dX}{dt} M_I \frac{dR_I}{dt} \tag{7.3}$$

and

$$W_T \frac{d^2 X}{dt^2} = \sum_I M_I (\frac{dR_I}{dt})^2 - 3N k_B T \tag{7.4}$$

where I represent each nucleus, M_I is the mass of each nucleus (see *amu*), R_I is the coordinate of each nucleus (see *xcart*), dX/dt is a dynamical friction coefficient, and T is the temperature of the thermostat (see *mditemp* and *mdftemp*).

7.11.27 ntime

Mnemonics: Number of TIME steps
Characteristic:
Variable type: integer parameter
Default is 5.

Gives the number of molecular dynamics time steps or Broyden structural optimization steps to be done if *ionmov*=1 or 2 respectively.

Note that at the present the option *ionmov*=1 is initialized with four Runge–Kutta steps which costs some overhead in the startup. By contrast, the initialisation of other *ionmov* values is only one SCF call.

ntime is ignored if *ionmov*=0.

7.11.28 ntypat

Mnemonics: Number of TYPEs of atoms
Characteristic: NO MULTI
Variable type: integer parameter
Default is 1.

Gives the number of types of atoms. E.g. for a homopolar system (e.g. pure Si) *ntypat* is 1.
The code tries to read the same number pseudopotential files.
The first pseudopotential is assigned type number 1, and so on ...

7.11.29 optcell

Mnemonics: OPTimize the CELL shape and dimensions
Characteristic:
Variable type: integer parameter
The Default is optcell=0

Allows to optimize the unit cell shape and dimensions, when *ionmov*=2 or 3. The configuration for which the stress almost vanish is iteratively determined, by using the same algorithms as for the nuclei positions. Will eventually modify *acell* and/or *rprim*. The ionic positions are ALWAYS updated, according to the forces. A target stress tensor might be defined, see *strtarget*

- *optcell*=0: modify nuclear positions, since *ionmov*=2, but no cell shape and dimension optimisation.
- *optcell*=1: optimisation of volume only (do not modify *rprim*, and allow an homogeneous dilatation of the three components of *acell*)
- *optcell*=2: full optimization of cell geometry (modify *acell* and *rprim* — normalize the vectors of *rprim* to generate the *acell*). This is the usual mode for cell shape and volume optimization. It takes into account the symmetry of the system, so that only the effectively relevant degrees of freedom are optimized.
- *optcell*=3: constant–volume optimization of cell geometry (modify *acell* and *rprim* under constraint — normalize the vectors of *rprim* to generate the *acell*)

- *optcell*=4,5 or 6: optimize *acell*(1), *acell*(2) or *acell*(3), respectively (only works if the two other vectors are orthogonal to the optimized one, the latter being along its cartesian axis).
- *optcell*=7,8 or 9: optimize the cell geometry while keeping the first, second or third vector unchanged (only works if the two other vectors are orthogonal to the one left unchanged, the latter being along its cartesian axis).

NOTE that a few details require attention when performing unit cell optimisation:

- one has to get rid off the discontinuites due to discrete changes of plane wave number with cell size, by using a suitable value of *ecutsm*;
- one has to allow for the possibility of a larger sphere of plane waves, by using *dilatmx*;
- one might have to adjust the scale of stresses to the scale of forces, by using *strfact*.
- if all the reduced coordinates of atoms are fixed by symmetry, one cannot use *toldff* to stop the SCF cycle. (Suggestion: use *toldfe* with a small value, like 1.0×10^{-10})

It is STRONGLY suggested first to optimize the ionic positions without cell shape and size optimization (*optcell*=0), then start the cell shape and size optimization from the cell with relaxed ionic positions.

Presently (v3.1), one cannot restart (*restartxf*) a calculation with a non–zero *optcell* value from the (x,f) history of another run with a different non–zero *optcell* value. There are still a few problems at that level.

7.11.30 restartxf

Mnemonics: RESTART from (X,F) history
Characteristic:
Variable type: integer parameter
Default is 0.

Control the restart of broyden minimisation.

Works only for *ionmov*=2 (Broyden) and when an input wavefunction file is specified, thanks to the appropriate values of *irdwfk* or *getwfk*.

If positive, the code reads from the input wf file, the previous history of atomic coordinates and corresponding forces, in order to continue the work done by the job that produced this wf file. If *optcell* $\neq 0$, the history of *acell* and *rprim* variables is also taken into account. The code will take into consideration the whole history (if *restartxf*==1), or discard the few first (x,f) pairs, and begin only at the pair whose number corresponds to *restartxf*.

If zero, the Broyden minimization is done from scratch.

NOTE: the input wf file must have been produced by a run that exited cleanly. It cannot be one of the temporary wf files that exist when a job crashed.

Presently (v3.1), one cannot restart a calculation with a non–zero *optcell* value from the (x,f) history of another run with a different non–zero *optcell* value. There are still a few problems at that level. Starting a non–zero *optcell* run from a zero *optcell* run should work.

7.11.31 signperm

Mnemonics: SIGN of PERMutation potential
Characteristic:
Variable type: integer
Default is 1.

In development. See the routine moldyn.f. See also *delayperm*.
+1 favors alternation of species
−1 favors segregation

7.11.32 strfact

Mnemonics: STRess FACTor
Characteristic:
Variable type: real parameter
Default is 100.0 (Bohr^2)

The stresses multiplied by *strfact* will be treated like forces in the process of optimization (*ionmov*=2, non–zero *optcell*).

For example, the stopping criterion defined by *tolmxf* relates to these scaled stresses.

7.11.33 strprecon

Mnemonics: STRess PRECONditioner
Characteristic:
Variable type: real parameter
Default is 1.0

This is a scaling factor to initialize the part of the Hessian related to the treatment of the stresses (optimisation of the unit cell). In case there is an instability, decrease the default value, e.g. set it to 0.1.

7.11.34 strtarget

Mnemonics: STRess TARGET
Characteristic:
Variable type: real array strtarget(6)
Default is 6*0.0 (Ha/Bohr**3)

The optimization of cell size and shape, as might be asked through *optcell*, will target the stress tensor defined by by *strtarget*, or part thereof (if restricted optimizations are asked, like fixed shape). Presently, this required target stress is not taken into account for the determination of the symmetries. If it breaks the symmetries of the input unit cell, so that *symrel* disagrees with *strtarget*, the result will be unreliable. Also, presently, the thermodynamical potential to be used n this situation (the free energy) does not replace the total energy, so that, for exemple, *ionmov*=3 cannot be used, since this algorithm is taking into account the total energy.

The components of the stress tensor must be stored according to: (1,1)→1; (2,2)→2; (3,3)→3; (2,3)→4; (3,1)→5; (1,2)→6. The conversion factor between Ha/Bohr^3 and GPa is: 1 Ha/Bohr^3 = 29421.033d0 GPa.

Not used if *optcell*==0.

7.11.35 tolmxf

Mnemonics: TOLerance on the MaXimal Force
Characteristic:
Variable type: real parameter
Default is 5.0d-5 hartree/bohr.

Sets a maximal absolute force tolerance (in hartree/bohr) below which BFGS structural relaxation iterations will stop.

Can also control tolerance on stresses, when *optcell* $\neq$ 0, using the conversion factor *strfact*. This tolerance applies to any particular cartesian component of any atom, excluding fixed ones. See the parameter *ionmov*.

This is to be used when trying to equilibrate a structure to its lowest energy configuration (*ionmov*=2).

A value of about 5.0d-5 hartree/bohr or smaller is suggested (this corresponds to about 2.5d-3 eV/Å).

No meaning for RF calculations.

7.11.36 vel

Mnemonics: VELocity
Characteristic: EVOLVING
Variable type: real array vel(3,*natom*)
Default is 3**natom* 0's.

Gives the starting velocities of atoms, in cartesian coordinates, in bohr/atomic time units (atomic time units given where *dtion* is described).

Irrelevant unless *ionmov* > 0.

For *ionmov*=8 (Nose thermostat), if *vel* is not initialized, a random initial velocity giving the right kinetic energy will be generated.

If the geometry builder is used, *vel* will be related to the preprocessed set of atoms, generated by the geometry builder. The user must thus foresee the effect of this geometry builder (see *objarf*).

Velocities evolve is *ionmov*==1.

7.11.37 vis

Mnemonics: VIScosity
Characteristic:
Variable type: real parameter
Default is 100.

Gives the viscosity (atomic units) for linear frictional damping term applied to molecular dynamics when *ionmov*=1. Used for eventual relaxation of structure (however, *ionmov*=2 is in general more efficient).

The equation of motion is:

$$M_I d^2R_I/dt^2 = F_I - vis dR_I/dt$$

The atomic unit of viscosity is hartrees*(atomic time units)/bohr^2. Units are not critical as this is a ficitious damping used to relax structures. A typical value for silicon is 400 with *dtion* of 350 and atomic mass 28 amu. Critical damping is most desirable and is found only by optimizing *vis* for a given situation.

7.11.38 wtatcon

Mnemonics: WeighTs for AToms in CONstraint equations
Characteristic: NO MULTI
Variable type: real array wtatcon(3,*natcon*,*nconeq*)
Default is 0.

Gives the weights determining how the motion of atoms is constrained during structural optimization or molecular dynamics (see *nconeq*, *natcon*, and *iatcon*). For each of the *nconeq* independent constraint equations, *wtatcon* is a 3**natcon* array giving weights, W_I, for the x, y, and z components of each of the atoms (labeled by I) in the list of indices *iatcon*. Prior to taking an atomic step, the calculated forces, F_I, are replaced by projected forces, F'_I, which satisfy the set of constraint equations

$$\sum_{mu=x,y,z;I=1,natcon} : W_{mu,I} * F'_{mu,I} = 0 \quad \text{for each of the } nconeq \text{ arrays } W_I.$$

Different types of motion constraints can be implemented this way. For example,

```
nconeq 1 natcon 2 iatcon 1 2 wtatcon 0 0 +1 0 0 -1
```

could be used to constrain the relative height difference of two adsorbate atoms on a surface (assuming their masses are equal), since $F'_{z,1} - F'_{z,2} = 0$ implies $z_1 - z_2 =$ constant.

Index