



Scientific Computing & Modelling

BAND User's Guide

ADF Program System

Release 2004.01

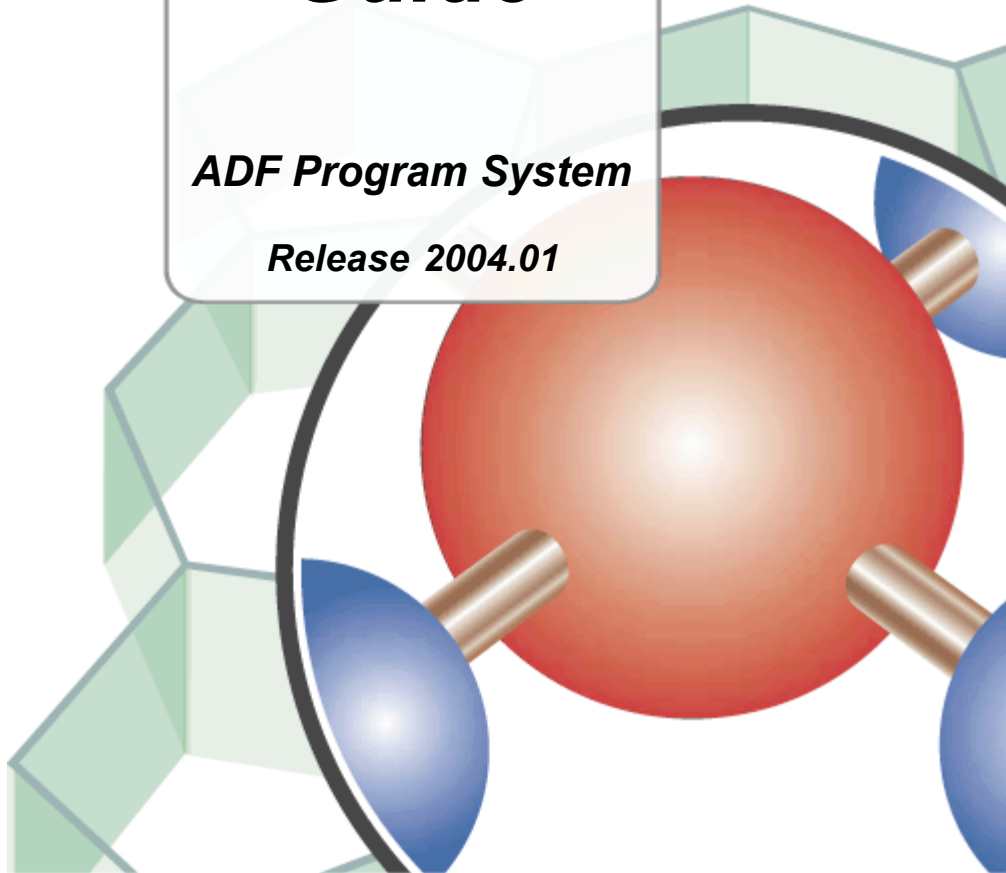


Table of Contents

BAND User's Guide

Preface

1 General

- 1.1 Introduction
 - Characterization of band

2 Input

- 2.1 Introduction
 - Format of the Input file
 - Units
 - Energy and Potential XC Functionals
 - Examples
 - Setting up an Input File
 - Numerical Atoms, Basis functions, and Fit functions
- 2.2 Main options
 - Energy Functional
 - Precision
 - Self-consistency
 - Restarts
 - Reference and Startup Atoms
 - Printed Output
 - Fragments
 - Time-dependent DFT
 - More Analysis

3 Recommendations, Problems, Questions

- 3.1 Recommendations
- 3.2 Trouble Shooting
 - Basis set dependency

4 Appendices

- 4.1 List of Keys
 - First List
 - Second List
 - Third List
- 4.2 Auxiliary Input Features

References

Preface

The installation of the Amsterdam Density Functional band-structure program (band) on a computer is explained in the Installation manual, where you can also find information about the license file(s) that you need to run the program. This User's Guide describes how to use the program, how input is structured, what files are produced, and so on. The Examples document, also available on the SCM web site, explains the most popular features in detail, by commenting on the input and output files in the \$ADFHOME/examples/band directory.

Where references are made to the operating system (OS) and to the file system on your computer the terminology of UNIX type OSS is used and a hierarchical structure of *directories* is assumed. The Mac OS X platform, Linux, and all popular Windows platforms are supported.

This guide and other documentation is available on the www via the Scientific Computing & Modelling home page at <http://www.scm.com>. As mentioned in the license agreement, it is mandatory, for publications in which BAND has been used, to cite the lead references given in the References document, which can be found on the SCM website.

Release 2002.03

Important new functionality has been added to BAND:

The time-dependent DFT implementation from Prof. Snijders group in Groningen is made available. This document describes the input options. The Examples document describes some example input and output files.

Some new options are made available that help to speed up large BAND calculations and reduce the amount of I/O that is required. The most relevant options are `CONFINEMENT`, `TAILS`, and `NONORTHOGONALSCFBASIS`. BAND users who wish to do more efficient BAND calculations without significant loss of accuracy are advised to study the relevant sections in this document, as well as the example described in the Examples document.

Some minor bugs have been fixed. The most important fix is that it is now (again) possible to do a BAND calculation without any numerical orbitals (only Slater orbitals).

Release 2000.02

The most important modifications introduced in version 2000.01 (and 2000.02) with respect to the 1999.x releases are:

A new fragment functionality has been implemented, for analysis purposes

Contrary to previous versions, the option to use abbreviated keywords (or extended keywords) has been disabled. This feature caused misunderstanding, confusion and hard-to-catch input mistakes, while it added little convenience. In version 2000.x, keywords must be typed completely and exactly.

The user will be warned when incorrect keywords are specified in input.

Most keywords are case insensitive now. Exceptions are of course where file or directory names are involved.

New XC functionals are available.

Input format has changed (end-of-key tokens and many keywords!)

1 General

1.1 Introduction

band is a program for calculations on periodic systems, i.e. polymers, slabs and crystals, and is supplemental to the molecular adf program for non-periodic systems. It employs density functional theory in the Kohn-Sham approach. BAND is very similar to ADF in the chosen algorithms, although important differences remain. Like ADF, BAND makes use of atomic orbitals, it can handle elements throughout the periodic table, and has several analysis options available.

Characterization of band

Functionality

- Formation energy with respect to isolated atoms that are computed with a fully numerical Herman-Skillman type subprogram
- A choice of density functionals, including LDA (Local Density Approximation) and GGA (Generalized Gradient Approximation) formulas
- Time-dependent DFT (available for the first time in 2002.03) for calculation of frequency-dependent dielectric functions of systems periodic in one or three dimensions.
- The ZORA method for scalar relativistic effects is available (also for the TDDFT option). Spin-orbit coupling can be taken into account.
- Automatic geometry optimization, dynamics, or other features requiring analytic gradients have not yet been implemented.

Analysis

- Mulliken populations for basis functions, overlap populations between atoms or between basis functions.
- Hirshfeld charge analysis
- Densities-of-States: DOS, PDOS and OPWDOS/COOP
- Form factors (X-ray structures)
- Charge analysis using Voronoi cells (yielding Voronoi Deformation Charges)
- Orbital plots
- One-electron energies and orbitals at the Brillouin Zone sample points
- Fragment orbitals and a Mulliken type population analysis in terms of the fragment orbitals

Technical

- SCF convergence based on a Direct Inversion of Iterative Subspace (DIIS) method
- The implementation is built upon a highly optimized numerical integration scheme for the evaluation of matrix elements of the Hamiltonian, property integrals involving the charge density, etc. This is the same numerical integration scheme as used in ADF.
- The program has been parallelized and vectorized
- Basis functions are Slater-Type Orbitals (STOs) and/or Numerical Orbitals (NOs).
- Fit functions are Slater-type exponential functions centered on the atoms and are used to fit the deformation density, which is the difference between the final density and the startup density. The deformation density has zero charge and will in general be small. The fitted deformation density is used for the calculation of the Coulomb potential and the derivatives of the total density (needed for the gradient corrections in the exchange-correlation functionals). In both cases the main part, due to the startup density, is calculated accurately by a numerical

procedure,

and only the small part from the deformation density is obtained via the fit.

- A frozen core facility is provided to allow efficient treatment of the inner atomic shells.
- Space group symmetry is used to reduce the computational effort in the integrals over the Brillouin zone.

2 Input

2.1 Introduction

When the program has been installed properly on your machine, you should be able to run it by supplying appropriate input. This document describes how input is organized. A few sample runs are contained in the distribution, see the Examples document on the SCM website for a description.

Input is structured by keywords, or keys. A key is a string of letters, dollar signs (\$), and digits. It must not start with a digit. It must not contain any other character, in particular not any blank.

Key-controlled input occurs in two forms: either one record (which contains both the key and/or associated data) or a sequence of records, collectively denoted as a key *block*. The first record of the block specifies the key (and may supply additional information); the block is closed by the 'end-key' code: a record containing only the word 'End'. The other records in the block provide data associated with the key.

The form to be used for a key is not optional: each admissible key corresponds to a particular form. The block form is used for keys that relate to 'lists' of data, such as atomic coordinates or basis functions.

As is the case in the molecular code, the 'title' must be specified explicitly with the key 'Title' and may be put on any record of the input file. The input file should end with a record `End Input`, that is, the program reads input until such a record is encountered or until the Fortran end-of-file condition becomes true, whichever comes first.

`End Input` is not a key.

Summarizing, the input file must have the following format:

Format of the Input file

```
TITLE jobname
  key-controlled data
  key-controlled data
  (et cetera...)
End Input
```

The ordering of keys in input is free and has no consequences.

All numerical information is 'free format': the absolute positions of numbers and the format of reals is irrelevant.

In most cases lowercase characters and capitals can both be used and the program will not discriminate between them, except in filenames.

Units

Geometric data (atomic positions and lattice vectors) are by default understood to be in atomic units (bohr). Alternatively one may supply data in angstroms by setting the key `units` (see below). Internally the program will then convert the input data into a.u.

Energy and Potential XC Functionals

The program calculates by default the self-consistent field solution from the potential in the spin-restricted local density approximation (LDA) to the exchange-correlation (XC) potential. The post-SCF generalized gradient (GGA) corrections for the exchange and the correlation energies (each for a few different functionals) are calculated and printed on output. The program can be instructed to use the GGA corrections in the potential during the SCF as well. This will in general have little effect on the formation energy, but it may affect the Density of States (DOS).

Examples

A few of examples are available in the \$ADFHOME/examples/band directory: for each sample run a run script and the output file. You may rerun the sample runs and compare your results with the provided output files to check that your band version has been installed correctly. This will also help you to get familiar with using the program. See the Examples document for a brief discussion of the available sample runs. The examples typically take a few minutes of CPU time.

For new users it may be convenient to have at least one of the example outputs at hand when reading the next parts, so that remarks and instructions can be compared to an actual case.

Setting up an Input File

We now give a brief guide to set up input.

You first specify the minimally required information: the geometric structure and the definition of the atoms (positions and function sets) that constitute the system. Then you add keys for all aspects for which you don't want to use the defaults.

1. Specify a title using the key `Title`.
2. Define the structure of periodicity with the key `Lattice`.
3. Give the positions of the atoms with the key `Atoms`.
All atoms specified in the corresponding data block belong to one *type* of atom.
Consequently, the key `Atoms` must be repeated as many times as there are different types of atoms.
Different chemical elements necessarily belong to different types.
The reverse is not true: it is allowed to define more than one 'type' of Carbon atom,
for instance to equip them with different basis sets.
4. For each of the types defined by the `Atoms` keys (and in the same order) create the `AtomType` block key.
This block key must contain the following
 - Define the numerical (Herman-Skillman-type) spherically symmetric free atom:
(block-type) key `Dirac`.
 - Add (optionally) Slater-type basis functions; key: `BasisFunctions`.
 - Supply fit functions (used for evaluation of the Coulomb potential from the charge density);
key: `FitFunctions`.
5. Add further keys for features you don't want to be defaulted, for instance
`Unrestricted` (if you want to do a spin-*unrestricted* calculation),
`Accuracy` (general precision parameter),
`KSpace` (parameter for numerical integration over the Brillouin Zone),
`DOS` (generation of the density-of-states), and so on.
6. Terminate the input file by typing `End Input`

Consult the sample runs and description below, to see how you may use the various keys (`Lattice`, `Atoms`,

AtomType).

Numerical Atoms, Basis functions, and Fit functions

The program starts with a calculation of the free atoms, assuming spherical symmetry. The formation energy is calculated w.r.t such atoms. You have to specify the configuration (i.e. which orbitals are occupied) in the `Dirac` subkey of the block key `AtomType`, and you can for instance use the experimental configuration. Keep in mind, however, that this is not necessarily the optimal configuration for your density functional. For instance, Ni has experimentally two electrons in the 4s shell, but with LDA you will find that it is energetically more profitable to move one electron from the 4s to the 3d. The configuration of the reference atoms does not (i.e. should not) affect the final (SCF) density.

Besides the available basis sets in `$ADFHOME/atomicdata/Band`, you could in principle use the basis functions from the database of the molecular ADF program (see the documentation of `adf` for how this database is organized). The functions you will find there are `STOs`, which is not optimal since `band` offers you the option to use `NOs` from the numerical atom. The most efficient approach is to use the `NOs` and remove from the `adf` basis set those `STOs` that are already well described by the `NOs`.

As an example we will construct a basis for the Ni atom with orbitals frozen up to the `2p` shell, derived from a triple-zeta `adf` basis. In the `Dirac` subkey of the block key `AtomType` you specify that the `NOs` up to `2p` should be kept frozen and that the `3d` and `4s` `NOs` be included in the valence basis. Copy from the `adf` database all `3d`, `4s` and the polarization functions into the `BasisFunctions` subkey of the block key `AtomType` and remove the middle `STOs` of the `3d` and the `4s`.

Usually it is already quite adequate for a good-quality basis to augment each `no` with one `STO`. You could then take a double zeta ADF basis and remove one of the `3d` and one of the `4s` `STOs`. We often find that such a basis, with one `STO` added per `NO`, has a quality that is comparable to *triple* zeta `STO` sets. We strongly recommend that you use combined `NO/STO` bases. Of course, you may want to verify the quality of the basis set by calculations on a few simple systems.

You can copy the fit functions from the ADF database into the `FitFunctions` subkey of the block key `AtomType`. As a matter of experience (and justified by a somewhat different handling of fit functions between the two programs), `BAND` is in most cases (we have not yet seen an exception) less sensitive to the quality of the fit set than `ADF` is.

2.2 Main options

This section is intended to introduce and thematically group together the keys listed in the next chapter.

Energy Functional

The energy functional consists of coulomb, kinetic and exchange-correlation (`xc`) terms. The starting point for the `xc` functional is usually the result for the homogeneous electron gas, after which so called nonlocal or generalized gradient corrections (GGA: Generalized Gradient Approximation) are added. See the keys `Spin` and `XC`.

Precision

The key `Accuracy` controls (amongst others) the accuracy of the numerical integration scheme (barring a few technicalities the same scheme as used in the molecular `adf` code).

If you want to fine-tune the integration scheme, see the description of the `Integration`

Self-consistency

The SCF procedure searches for a self-consistent density. The self-consistent error is the square root of the integral of the squared difference between the input and output density of the cycle operator. When the SCF error is below a certain criterion, controlled by subkey `Criterion` of block key `Convergence`, convergence is reached. In case of bad convergence the SCF search can be controlled by the `DIIS` block, of which the most interesting entry is `Potential`, implying that the `DIIS` scheme is applied to the potential, rather than the (valence) density.

Restarts

The DOS section that outputs information of the SCF solution can be executed with key `Restart`, subkey `DOS` and orbital plots can be obtained with subkey `orbitalplot` (note: currently disabled!). In the future subkey `SCF` will make it possible to continue a (unconverged) SCF procedure.

Reference and Startup Atoms

The formation energy of the crystal is calculated with respect to the reference atoms. `band` gives you the formation energy with respect to the spherically symmetric spin-*restricted* LDA atoms. If you want the program to do the spin-unrestricted calculation for the atoms you can give key `Unrestricted` the extra option `Reference`. We do not recommend this as it would give you the false (except in special cases) feeling that you've applied the right atomic correction energy so as to obtain the 'true' bonding energy with respect to isolated atoms. The true atomic correction energy is the difference in energy between the used artificial object, i.e. the spherically symmetric, spin-restricted atom with possibly fractional occupation numbers, and the appropriate multiplet state. The spin-*unrestricted* reference atom would still be spherically symmetric, with possibly fractional occupations: it would only have the probably correct (Hund's rule) net spin polarization.

The startup density is normally the sum of the restricted atoms. In case you do an unrestricted calculation you may want to get the sum of the unrestricted atoms as startup density by giving key `Unrestricted` the extra option `Startup`. This does not always provide a better startup density since all atoms will have their net-spins pointing up. If a frozen core is used this option can sometimes lead to a negative valence density, because the frozen core is derived from the restricted atom. The program will stop in such a case.

No matter what reference or startup atoms you use, core orbitals and NOs originate always from the restricted free-atom calculation, because we don't want a spatial dependence of the *basis functions* on spin.

Printed Output

The amount of output in different stages of the program is controlled by print keys, that can be toggled with the key `Print` followed by a key (see description of `Print` key).

Fragments

Unlike the ADF program `BAND` is not based on fragments. Quite fundamentally the building blocks are the atoms. A fragment feature is available albeit rather primitive. A typical application is the periodical adsorption of one or more molecules on a surface. For instance, consider periodic adsorption of hydrogen molecules

over a surface. First you calculate the free molecule in the same orientation as when adsorbed to the substrate and obtain the fragment orbitals with the option

```
| BASIS
  PREPAREFRAGMENT
| END
```

Since you would like to use a molecular fragment, it makes sense to put the molecules far apart (large lattice spacing) and force dispersion to be neglected (KSPACE 1). To use the fragment you need the result file of this calculation, therefore specify

```
| SAVE RUNKF
```

somewhere in the input, and rename RUNKF to the name of the result file.

Specifying

```
| PRINT EIGENS
```

for this calculation produces output concerning the eigen states, thereby providing a means to identify the eigen states (e.g. to be sigma, pi, et cetera).

Next, prepare the input for the over-layer with the substrate. There are two restrictions: the atoms of the fragment (the hydrogens in the example) should be the first atoms on the input list and they should not be of the same type. You can force atoms of the same chemical element to be of a different type, by putting them in a different `Atoms` block.

You make a second `Atoms` block for the second hydrogen. Of course, you have to add also a `AtomType` block. Tell the program that the two hydrogens constitute a fragment by

```
| NATOMSASFRAGMENT 2
```

Finally, you add to the input the `Fragments` key block where you give the name of the fragment file, and the subkey `Simplefrag` of key `Basis`, which toggles the molecular fragment option. The basis will be transformed accordingly, and also the partitioning of basis functions is affected. The new basis and partitioning are reflected in the Mulliken gross and overlap populations. It is allowed to have more than one fragment. The key `FragmentLabels` gives you the possibility to introduce labels for the fragment orbitals.

An example of using the fragments feature in BAND is provided in one of the sample runs (CO on a Cu surface) in the directory `$ADFHOME/examples/band/e_Frags_COCu`, see the Examples document.

The use of fragments is a valuable analysis tool. A previous fragment option in ADF-BAND could only be used for restart purposes. The new fragment option provides the possibility to use molecular fragments in

a subsequent periodic calculation. The provided example is a slab calculation of Cu with an CO molecule adsorbed. A DOS analysis is performed in terms of the Cu atomic orbitals and the CO molecular orbitals.

The new `Basis` subkey `Simplefrag` option (key) uses a molecular fragment that is created in the first part of this sample run. Note the `Basis` subkey `PrepareFragment` and the use of `KSPACE 1`, together with a large lattice spacing. In this way a 'molecular' solution is obtained, which can be used as a fragment.

This fragment has been saved as `t21.CO`, and is input for the second step of this example. The basis is transformed according to the eigenvectors of the CO fragment. Orbital labels are adapted by specifying `FragmentLabels`.

The Density-of-States outputs are computed in the adapted basis.

The results can be interpreted in terms of the molecular solutions of CO, making the analysis easier.

Time-dependent DFT

This feature has become available for the first time in ADF2002.03.

In this section, the time-dependent density functional theory implementation in ADF-BAND is described. It describes how to do a response calculation in ADF-BAND (which input keys to use). The TDDFT module enables the calculation of real and imaginary parts of the material property tensor $\chi_e(\omega)$ called the electric susceptibility, and the macroscopic dielectric function $\epsilon_e(\omega)$. These are mutually related,

$$\epsilon_e(\omega) = 1 + 4\pi\chi_e(\omega).$$

In general $\chi_e(\omega)$ and $\epsilon_e(\omega)$ are tensors, which, however, simplify to scalars in isotropic systems. Some examples are available in the \$ADFHOME/examples/band directory and are discussed in the Examples document.

References

Background on the implementation and examples of previous applications of this method can be found in the following references. Kootstra's Ph.D. thesis contains much background information. It can be obtained from the SCM website.

- [1] F. Kootstra, P. L. de Boeij, and J. G. Snijders, J. Chem. Phys. 112, 6517 (2000).
- [2] F. Kootstra, P. L. de Boeij, and J. G. Snijders, Phys. Rev. B 62, 7071 (2000).
- [3] F. Kootstra, P. L. de Boeij, H. Aissa, and J. G. Snijders, J. Chem. Phys. 114, 1860 (2001).
- [4] P. L. de Boeij, F. Kootstra, and J. G. Snijders, Int. J. Quantum Chem. 85, 449 (2001).
- [5] P. L. de Boeij, F. Kootstra, J. A. Berger, R. van Leeuwen, and J. G. Snijders, J. Chem. Phys. 115, 1995 (2001).
- [6] F. Kootstra, P. L. de Boeij, R. van Leeuwen, and J. G. Snijders, Festschrift in honour of R. G. Parr, Editor K. D. Sen, accepted.
- [7] F. Kootstra, P. L. de Boeij, and J. G. Snijders, J. Chem. Phys. to be submitted.
- [8] F. Kootstra, P. L. de Boeij, R. van Leeuwen, and J. G. Snijders, to be submitted.
- [9] F. Kootstra, Ph.D. thesis, Rijksuniversiteit Groningen, Groningen (2001).

Description of TDDFT Input options

The key to use in the input file is RESPONSE if you want to perform a periodic TDDFT (response) calculation. This key is optional. It is a block type key.

```
RESPONSE
  nfreq  5
  strtfr 0.0
  endfr  0.01
  cnvi   0.001
  cnvj   0.001
  ebndt1 0.001
  ifx    0
  isz    0
  iyxc   0
END
```

Omitting the specific options in the RESPONSE block will cause default setting to be used during the calculation, as given above.

`nfreq` the number of frequencies in a.u. for which a TDDFT calculation is performed when calculating the dielectric function $\epsilon_e(\omega)$ of a system (default=5).

`strtfr` is the start frequency in a.u. of the frequency range over which the dielectric function is calculated (default=0d0).

`endfr` is the end frequency in a.u. of the frequency range over which the dielectric function is calculated (default=1d-2).

`cnvi` the first convergence criterium for the change in the fitcoefficients for the fitfunctions, when fitting the density (default=1d-3).

`cnvj` the second convergence criterium for the change in the fitcoefficients for the fitfunctions, when fitting the density (default=1d-3).

`ebndt1` the energy band tolerance, for determination which routines to use for calculating the numerical integration weights, when the energy band posses no or to less dispersion (default=1d-3).

`ifxc` integer indicating which fxc kernel is used (default=0).

0 = Adiabatic Local Density Approximation (ALDA) (Can. J. Phys. 58, 1200 (1985)).

1 = Gross-Kohn, frequency dependent fxc kernel (PRL 55, 2850 (1985), 57, 923 (1986)),

2 = van Leeuwen-Baerends (LB94) (PRA 49, 2421 (1994))

`isz` integer indicating whether or not scalar zeroth order relativistic effects are included in the TDDFT calculation (default=0).

0 = relativistic effects are not included,

1 = relativistic effects are included.

`iyxc` integer for printing yxc-tensor (default=0) (JCP 115, 1995 (2001)).

0 = not printed,

1 = printed

More Analysis

If you are interested in the DOS or a partitioning of the DOS or a partitioning of the DOS, look for the description of the keys `DOS`, `GrossPopulations` for partial DOS, and `OverlapPopulations` for overlap DOS.

The eigen system can be printed with `PRINT EIGENS` and the Mulliken populations per orbital per k-point can be printed with `PRINT ORBPOP`.

A list of the basis functions will be printed with `PRINT ORBLABELS`.

3 Recommendations, Problems, Questions

3.1 Recommendations

Use numerical orbitals (from the DIRAC internal subprogram) and add STOs for increased flexibility. This turns out to be more (although not by much) rapidly converging to the basis set limit than when you use STOs only, in particular for relatively small basis sets.

3.2 Trouble Shooting

Basis set dependency

A calculation aborts with the message: dependent basis. It means that for at least one k-point in the BZ the set of Bloch functions, constructed from the elementary basis functions (atom-centered and plane waves, if applicable) is so close to linear dependency that the numerical accuracy of results is in danger. To check this, the program computes, for each k-point separately, the overlap matrix of the Bloch basis (normalized functions) and diagonalizes it. If the smallest eigenvalue is zero, the basis is linearly dependent. (Negative values should not occur at all!). Given the limited precision of numerical integrals and other aspects in the calculation, you are bound for trouble already if the smallest eigenvalue is very small, even if not exactly zero. The program compares it against a criterion that can be set in input (key `Dependency` option `Basis`).

If you encounter such an error abort, you are strongly advised not to adjust the criterion so as to pass the internal test: there were good reasons to implement the test and to set the default criterion at its current value. Rather, you should adjust your basis set. Generally speaking, you should remove one or more basis functions and maybe modify some of the (other) STO basis functions. The program prints information that helps you determine which basis functions should be modified/removed. Another way to modify your basis set, is to use the confinement keyword. This has the effect of making the diffuse basis functions more localized, thus reducing problematically large overlap with similar functions on neighboring atoms.

In the standard output file, after the error message, you will find a list of eigenvalues of the overlap matrix. If only the first is smaller than the threshold, you should remove one basis function. If more eigenvalues are very small, it is likely that you have to remove more than one function, although you can of course try how far you can get by eliminating just one.

Next the program prints the so-called Dependency Coefficients: a list of numbers, one for each basis function. Those with a large value are the suspicious ones. If you find two coefficients that are significantly larger than the others, you should replace the two corresponding functions by one. Easiest is to remove one of them (take the one with the bigger coefficient). If one of them is a numerical orbital from Dirac and the other an STO, remove the STO. If both are STOs, remove one and replace the other by some kind of average (regarding the radial characteristic: exponential factor and power of radial coordinate).

To identify how the functions in your input correspond to the list the underlies the series of Dependency Coefficients, you have to set up the list of basis functions as follows:

- Consider an outer loop over all atom TYPES. These correspond, in order as well as in number, to the sequence of `AtomType` keys in your input file.
- For each type, consider a loop over all atoms of that type, i.e. the atoms in the ATOM block corresponding to the `AtomType` key at hand.
- For each atom (each `AtomType` key), first write down all DIRAC basis functions, then all STOs.

When writing down the functions, be aware that each entry in your input file specifies a function set, by the quantum number L and hence corresponds to $2L+1$ actual basis functions.

- Regarding the DIRAC basis functions: they belong to the list of basis functions only if the key `valence` occurs in the pertaining DIRAC input block. If not, no DIRAC functions of that type are included in the basis.

If the Dirac functions are included, you must omit the Core functions and include only the Valence functions

from that DIRAC block. The first record in your DIRAC block with two numbers defines (by the first number)

the total number of function sets in the DIRAC block (which you can verify by simple counting) and (by the second number) the number of Core function sets among them.

The Core function sets, if any, are always the first so many in the list in the DIRAC block.

The program stops as soon as it encounters a dependency problem. This may happen for the first k -point. After you have adjusted the basis set following the above guidelines, you will have solved it. However, it may easily happen that the problem shows up again, but now for another (later) k -point, where other entries in the basis set may cause trouble. Do not think you have repaired the first problem incorrectly. Just repeat the procedure until you pass all k -points in the basis set construction without errors. Typically (as a last remark), although not necessarily, the first k -point may have a dependency problem from too many s -type functions, while other k -points may be more sensitive to the series of p -functions in your basis.

4 Appendices

4.1 List of Keys

Next follows the list of keys with an explanation of their meaning and the appropriate way to use them. Most keys are of the simple (record-) type. Block type keys are indicated in the list by the phrase 'block type' between brackets after the key.

For the large majority of keys, their occurrence in input is optional. Omission leads to default values and default settings. A few keys *must* be given however, so as to define the system to be computed. These keys are `Title`, `Lattice`, `Atoms` and `AtomType` containing the `Dirac` and `FitFunctions` subkeys; they specify respectively the title of the job, the Bravais lattice, the nuclear positions and basic characteristics of the atom type, including the auxiliary (fit) functions used to compute the Coulomb potential.

If you inspect the source code you may notice that there are in fact more keys defined in the program than those enumerated in the list below. With an eye on future developments we have omitted in this documentation a discussion of keys that will be removed or changed *and* that have not much practical importance anyway.

The first list of keys below contains the most important ones. The second contains keys that you will probably not use very frequently as they relate to debugging, tests, and data organization inside the program, rather than to physical or methodological aspects. Most keys in this section are block keys that contain fine-tuning options for certain aspects of the code (e.g. screening, k-space integration, SCF behavior). A third list is added. This contains only undocumented (currently existing) development/debug keys and keys that existed in the previous version, but cannot be used anymore in ADF-BAND 2002.03 and later (obsolete keys).

First List

[ACCURACY](#) • [ATOMS](#) • [ATOMTYPE](#) • [BASIS](#) • [COMMENT](#) • [CONVERGENCE](#) • [COORDINATES](#) • [DEFINE](#) • [DIIS](#) • [DOS](#) • [FORMFACTORS](#) • [FRAGMENTS](#) • [FRAGMENTLABELS](#) • [GROSSPOPULATIONS](#) • [IGNORE](#) • [KLABELS](#) • [KSPACE](#) • [LATTICE](#) • [MAXMEMORYUSAGE](#) • [NATOMSASFRAGMENT](#) • [ORBITALLABELS](#) • [ORBITALPLOT](#) • [OVERLAPPPOPULATIONS](#) • [PLANEWAVES](#) • [PRINT](#) • [RELATIVISTIC](#) • [RESPONSE](#) • [RESTART](#) • [SCF](#) • [STOPAFTER](#) • [TAILS](#) • [TITLE](#) • [UNITS](#) • [UNRESTRICTED](#) • [XC](#)

ACCURACY

General precision parameter. Default value = 3.5. High values (larger than 4.5 say) should only be used in extreme cases. Values below 3.0 produce unreliable results, due to the limited precision in integrals. Accuracy plays in fact the role of a very general accuracy parameter. It determines not only the generation of integration points (it is the default of the `AccInt` key for the integration scheme), but also the (default) values of many other parameters and settings that are related to the accuracy of the results.

ATOMS (block-type)

Nuclear coordinates. The chemical symbol (H, C, Cu, etc.), which defines the atom *type*, must be given on the keyword-line. The data-records contain the coordinates, one atom per line. The coordinates are in cartesian representation unless the key `Coordinates` has been given in input with the value `natural`, in which case the numbers are interpreted as expansion coefficients in the Bravais lattice vectors (see key `Lattice`). In case of the cartesian representation the values are in atomic units (angstroms if the unit of length has been changed with key `Units`). The `atoms` key must occur once for every atom type, i.e. the number of atom types is *defined* as the number of occurrences of this key.

ATOMTYPE (block-type)

Description of the atom type. Contains the block keys `Dirac`, `BasisFunctions` and `FitFunctions`. The key corresponds to one atom type. The ordering of the `AtomType` keys (in case of more than one atom type) is not arbitrary. It is interpreted as corresponding to the ordering of the `Atoms` keys. The n -th `AtomType` key supplies information for the numerical atom of the n^{th} type, which in turn has atoms at positions defined by the n^{th} `Atoms` key.

```

ATOMTYPE Symbol
  DIRAC ChemSym
    {option}
    ...
    shells cores
    shell_specification {occupation_number}
    ...
  SUBEND
  {BASISFUNCTIONS
    shell_specification STO_exponent
    ...
  SUBEND}
  FITFUNCTIONS
    shell_specification STO_exponent
    ...
  SUBEND
END

```

The argument `symbol` to `AtomType` is the symbol that is used in the `Atoms` key block.

Dirac (block-type)

Specification of the numerical ('Herman-Skillman') free atom, which defines the initial guess for the SCF density, and which also (optionally) supplies Numerical Atomic Orbitals (NOs) as basis functions, and/or as fit functions for the crystal calculation. The argument *ChemSym* of this option is the chemical symbol of the atom type.

The data records of the `Dirac` key are: 1.the number of atomic shells (1s,2s,2p,etc.) and the nr. of core-shells (two integers on one line). 2,3 specification of the shell and its electronic occupation. This specification can be done via quantum numbers or using the standard designation (e.g. '1 0' is equivalent to '1s'). Optionally one may insert anywhere in the `Dirac` block a record `Valence`, which signifies that all numerical valence orbitals will be used as basis functions (NOs) in the crystal calculation.

You can also insert `NumericalFit` followed by a number (max. l-value) in the key block, which causes the program to use numerical fit functions. For example `NumericalFit 2` means that the squares of all s,p, and d NOs will be used as fit functions with l=0, since the NOs are spherically symmetric.

If you insert `Spinor`, a spin-orbit relativistic calculation for the single-atom will be carried out.

The Herman-Skillman program generates all its functions (atomic potential, charge density, one-electron states) as tables of values in a logarithmic radial grid. The number of points in the grid, and the min. and max. r-value are defaulted at 3000, 0.000001, and 100.0 (a.u.) respectively. These defaults can be overwritten by specifying anywhere in the `dirac` block the (sub)keys `radial`, `rmin` and `rmax`.

The program will do a spin-unrestricted calculation for the atoms in addition to the restricted one. The occupation of the spin-orbitals will be of maximum spin-multiplicity and cannot be controlled in the `DIRAC` key-block.

Basisfunctions (block-type)

Slater-type orbitals, specified by quantum numbers n,l or by the letter designation (e.g. 2p) and one real (alpha) per sto. One sto per record. Use of this key is optional in the sense that Slater-type functions are not needed if other basis functions have been specified (In the first place: numerical atomic orbitals, see key `DIRAC`. In the second place (only in bulk crystals): plane waves, see the key `planewaves`)

FitFunctions (block-type)

Slater-type fit functions, described in the same way as in `basisfunctions`. Each `fitfunctions` key corresponds to one atom type, the type being the one of the preceding `DIRAC` key.

The selection choice of a 'good' fit set is a matter of experience. Fair quality sets are included in the database of the molecular program `adf`.

Example:

```

ATOMTYPE C :: Carbon atom
  DIRAC C
    3 1
  VALENCE
    1s
    2s
    2p 2.0
  SUBEND
  BasisFunctions
    1s 1.7
    ...
  SubEnd
  FitFunctions
    1s 13.5
    2s 11.0
    ...

```

```

SubEnd
END

```

Remarks

Other subkeys of the `AtomType` key block are `CoreFunctions` and `TestFunctions` that have the same format as the `BasisFunctions` and `FitFunctions` blocks. The `TestFunctions` block specifies STOs to be used as test functions in the numerical integration package. The `CoreFunctions` key specifies the use of Slater type auxiliary functions to enforce that valence basis functions are orthogonal to all core orbitals. By default this is achieved by explicitly (numerically) projecting out the component of the core orbitals from the valence functions. This key is currently disabled.

BASIS (block-type)

Block key containing options and thresholds related to the basis set. See also second list.

Preparefragment

Needed to make a fragment (input) file.

Simplefrag

Fragment option that uses molecular fragments. The basis is actually transformed to the fragment basis. Keys `Fragments` and `NatomsAsFragment` are required.

COMMENT (block-type)

The content of this key is a text that will be copied to the output header, where general program information is also printed.

CONVERGENCE (block-type)

All options and parameters related to the convergence behavior of the SCF are defined in this block key. Also the finite temperature distribution is part of this. See also second list.

Criterion

Criterion for termination of the SCF procedure. Default depends on `Accuracy`, for instance 3e-4 for `Accuracy 4.0`

Allbands

Requires a numerical argument, which is an energy width, in a.u.. It simulates a finite-temperature electronic distribution. By default, zero temperature is assumed. The key may be used to achieve convergence in an otherwise problematically converging system (slabs, typically). The energy of a finite-T distribution is different from the T=0 value, but for small T a fair approximation of the zero-T energy is obtained by extrapolation. The extrapolation energy correction term is printed with the survey of the bonding energy in the output file. Check that this value is not too large. Build experience yourself how different settings may affect the outcomes. Remember that this key is meant to help you overcome convergence problems, not to do finite-T research: only the electronic distribution is computed T-dependent, other aspects are not accounted for!

Degenerate

Smooths (slightly) occupation numbers around the Fermi level, so as to insure that nearly-degenerate states get (nearly-) identical occupations. Default=off, but in case of problematic SCF convergence the program will turn this key on automatically, unless the key `Nodegenerate` is set in input. The smoothing depends on the argument to this key, which can be considered a 'degeneration width'. When the argument reads 'default', the program will use a default value for the energy width (1e-4).

COORDINATES

The only sensible value for this key is: `natural`, which specifies that nuclear coordinates (key atoms) are given as expansions in the Bravais lattice vectors, rather than in a Cartesian representation.

DEFINE (block-type)

Definition of user-supplied functions and variables that can subsequently be used in the input file. (see the note on auxiliary input features)

DIIS (block-type)

Most interesting option is `DiMix`. See second list.

DOS (block-type)

General Density-Of-States information.

```
DOS
  { File filename }
  { Energies n }
  { Min emin }
  { Max emax }
End
```

Where *n* is the number of (equidistant) energy-values, *emin*, *emax* the minimum and maximum energy values (with respect to the Fermi energy), and *filename* the (formatted) file on which the DOS-information will be written. If the file is omitted, the information will be printed in the output file.

Example:

```
DOS
  FILE      plotfile
  ENERGIES  500
  MIN       -.35
  MAX       1.05
END
```

According to this example, density-of-states values will be generated in an equidistant mesh of 500 energy values, ranging from 0.35 below the Fermi level to 1.05 above it (atomic units). All information will be written to a file *plotfile*. The information on the plot file is a long list of pairs of values (energy and DOS), with some informative text-headers and general information; since the file is formatted one can easily inspect its contents; it should be suitable for graphical software like *IGOR*.

Density-of-states values are generated for the *total* D.O.S. and optionally also for some partial densities of states (see the keys `GrossPopulations` and `OverlapPopulations`).

If the key `DOS` is omitted, no Density-of-States information is generated at all. If the key is given, but one or more of the 'subkeys' (file, energies, min, max) are omitted, default values are inserted (: direct printing on output, 300, -0.75, 0.75).

FORMFACTORS

X-ray structure factors (Fourier analysis of the charge density) are computed after termination of the SCF procedure; the key should be followed by an integer specifying the number of stars of K-vectors for which the structure factors are computed. Default (omission of the key): 3 stars.

FRAGMENTS (block-type)

Define fragments. This key takes as argument the fragment file name (absolute path or path relative to the executing directory) and its contents are for each atom in the fragment two integers: atom number in the fragment versus atom number in this calculation.

FRAGMENTLABELS (block-type)

The program will generate labels for the fragment orbitals automatically by default. With this option you can assign your own label to each fragment orbital.

Example:

```
FRAGMENTLABELS
  Sigma
  Sigma*
  Pi_x
  Pi_y
  Pi_x*
  Pi_y*
End
```

In this example the first four fragment orbitals will be labeled as stated in the body of this key. The remaining orbitals are labeled by the default labeling system (e.g. 1/FO/5, etc.). The labels are used in combination with options like PRINT EIGENS and PRINT ORBPOP. (See also PRINT ORBLABELS).

This key can be given once for each fragment.

GROSSPOPULATIONS (block-type)

Partial densities-of-states are generated for the gross populations listed under this key.

```
GROSSPOPULATIONS
  { iat lq }
  { FragFun iat ifun }
  { Frag kat }
  { Sum
    ...
    EndSum }
End
```

Each line contains a PDOS instruction. There are three possibilities:

1 Line contains two integers, the first specifying the atom (*iat*) (numbered according to the total list comprised from all atoms-keys), the second the *l*-value (*lq*) (0:s, 1:p, 2:d, and so on). Partial densities of states are generated for all real spherical harmonics belonging to the specified *l*-value.

2 Line is of the form: `Frag kat`, which means that the PDOS of the functions belonging to atom *kat* will be calculated.

3 Line is of the form `fragfun iat ifun`, which refers to the function *ifun* of atom *iat*, including core states of that atom.

You can sum PDOS commands with a sum-block, i.e. specify any number of any of the three PDOS specifications in a block that starts with `Sum` and ends with `EndSum`.

Example:

```
GROSSPOPULATIONS
FRAGFUN 1 2:: Second function of first atom
FRAG 2 :: Sum of all functions from second atom
SUM:: sum following PDOSes
  FRAG 1::Atom nr.1
  FRAGFUN 2 1::First function of second atom
  5 1:: All p functions of fifth atom
```

```

|   ENDSSUM
|   END

```

IGNORE (block-type)

Suppress reading of input until the next end-key code (END).

KLABELS (block-type)

With this key you assign names to symmetry unique k-points. In general you do not know in advance how many k-points will be used, nor the number of points that are symmetry unique among them. Therefore you should first add to your input file the option `stopAfter gemtry`, which causes the program to stop after the k-points have been generated. In the output you will see under the header `k-space` integration a numbered list of the k-points, with their, symmetry-unique-index number, and coordinates. In the next run you may then give the symmetry unique k-points symbolic names.

Example:

```

| KLABELS
|   GAMMA
|   X
|   M
| END

```

According to this example all k-points with symmetry unique number 1, will be labeled `gamma`, those with symmetry unique number 2 `X`, and those with symmetry unique number 3 `M`.

KSPACE

Parameter for numerical integration over the Brillouin Zone (k-space). An integer value should be supplied. 1=absolutely minimal (only the G-point is used), 2=linear tetrahedron method, coarsest possible spacing, 3=quadratic tetrahedron method, coarsest spacing. Higher values should be chosen odd (5, 7,...) to use the *quadratic* method; even values (4,6,...) trigger the linear tetrahedron method, which is usually by far inferior. cpu-time increases very rapidly with higher `k-space`-values: try 3 for a reasonable result, or 5 for higher precision. The key may occur as a block key, the contents of the key are options that handle in more detail the integration of the Brillouin zone. E.g. the subkey `kinteg` takes over the simple key `k-space`. For other settings, see the explanation of `k-space` in the second list.

Default depends on `Accuracy`, the integration parameter in real space.

LATTICE (block-type)

Vectors defining the Bravais lattice, one vector per line. The number of lines defines the periodicity of the system: 1: polymer, 2: slab, 3:bulk crystal. Each vector has three coordinates (Cartesian), in atomic. To specify coordinates in Angstrom, use the `Units` key.

MAXMEMORYUSAGE

The maximum amount of memory which may be allocated by workspace manager of the program, in Megabytes. Note that the program needs more memory than you specify here, e.g. due to the use of Fortran 90 allocates and I/O buffering.

NATOMSASFRAGMENT

This key is followed by a number `n`, signifying that the first `n` atoms are to be treated as a fragment. Your input file must contain the `fragments` blocks. You can have more than one fragment, in which case this key should be followed by a series of numbers, `n1 n2...`, which means that the first fragment consists of the first `n1` atoms, the second fragment of the `n2` next atoms, etc. Now the file should contain the `fragments` blocks of the first fragment, followed by the `fragments` blocks of the second fragment...

ORBITALLABELS (block-type)

The program will generate labels for the valence basis states automatically by default. With this option you can assign your own label to each valence function.

Example:

```
ORBITALLABELS
  SIGMA
  2P_Y
  2P_Z
  2P_X
END
```

In this example there are four valence basis functions. The first will be labeled SIGMA, the second 2P_Y and so on. The labels are used in combination with options like `print eigens` and `print orbpop`. (See also `print orbitallabels`).

Description of the automatic labels for the valence basis. This basis contains core functions needed for orthogonalization on the core. A normal atomic basis function, i.e. a numerical orbital or a Slater type orbital, gets a label like

<atom number>/<element>/<orbital type>/<quantum numbers description>/<exp in sto>

Example with a Li and a H atom:

```
1/LI/NO/1s
1/LI/NO/2s
1/LI/STO/2s/1.4
1/LI/STO/2p_y/1.3
1/LI/STO/2p_z/1.3
1/LI/STO/2p_x/1.3
2/H/NO/1s
2/H/STO/1s/1.9
...
```

Core states and plane waves will just get simple numbers as labels:

```
CORE STATE 1
CORE STATE 2
PW1
PW2
```

ORBITALPLOT (block-type)

Currently disabled.

With a result file of a previous calculation you may rerun the program, using that file for `restart`, to make a plot file of the real part of the eigenvectors in a plane.

Only eigenstates are treated that belong to bands which fall (at least partially) in the energy range defined by the `DOS` key. Therefore, the `orbitalplot` key must be used in conjunction with the `DOS` key.

The first four records in this key-block describe the set of plot points. The first record is the starting point (three coordinates). The second record should contain a vector, relative to the starting point, and a step size. The same holds for the third record. The plot points are generated in a plane spanned by these two vectors. The fourth record determines the amount of steps that are taken in the two directions. A plot point must not coincide with the position of an atom, because that would give rise to a singularity in the potential evaluation. If this happens, you have to displace the starting point.

You control which eigenvectors are used by a next series of records. Each such record contains the index of a k-point in the list of all symmetry unique k-points, and two integers that specify the range of bands to be used.

Example:

```

ORBITALPLOT :: Make plot file of eigenvectors
  0. 0. 0. :: Starting point
  1. 0. 0. .1 :: Vector1 and step1
  0. 1. 0. .1 :: Vector2 and step2
  10 10 :: N1 and n2
  2 1 5 :: Make plots for k=2 from band 1 to 5
  4 1 1 :: Make plot for k=4 and band =1
  ...
  ...
END

```

The resulting 'plot data' are printed in the standard output file, in the format:
x y value.

X and y are relative coordinates in the selected plane, with the origin defined as the lower-left corner, and value is the orbital value. All data for a particular orbital are contiguous, followed by the data for the next orbital, et cetera.

You can make a plot file of the basis functions, instead of eigenvectors, by inserting the option: `plotprimitive` into the key-block header.

Example:

```

ORBLOT PLOTPRIMITIVE :: Make plot file of basis functions
.....
END

```

OVERLAPPOPULATIONS (block-type)

Overlap population weighted densities-of-states are generated for the overlap populations listed

```

OVERLAPPOPULATIONS
  Left
    { iat lq }
    { FragFun jat ifun }
    { Frag kat }
  Right
    ...
End

```

You can use this to get the overlap population weighted densities-of-states (OPWDOS), also known as the crystal orbital overlap population (COOP), of two functions, or, if you like, one bunch of functions with another bunch of functions. The key-block should consist of left-right pairs. After a line with `left` you enter lines that specify one or more functions (see `GrossPopulations`), followed by a similar structure beginning with `right`, which will produce the OPWDOS of the left functions with the right functions.

Example:

```

OVERLAPPOPULATIONS
  LEFT::First OPWDOS
    FRAG 1
  RIGHT
    FRAG 2
  LEFT:: Next OPWDOS
    FRAGFUN 1 1
  RIGHT
    2 1
    FRAGFUN 3 5

```

| END

PLANEWAVES

warning: currently *disabled!*

The integer after this key specifies the number of stars of K-vectors, defining plane wave basis functions that will be used in the crystal valence basis. Can be used only in bulk crystals. Default: no plane waves at all, only Atomic Orbitals (numerical from the Herman-Skillman program, and Slater-type, via the subkey Basisfunctions of key AtomType)..

PRINT

One or more strings (separated by blanks or comma's) from a pre-defined set may be typed after the key. This induces printing of various kinds of information, usually only used for debugging and checking. The set of recognized strings frequently changes (mainly expands) in the course of software-developments. Useful arguments may be `symmetry`, and `fit`. A list of all important arguments to this key follows:

EIGENS

Prints the (complex) coefficients in the form (norm, phase factor) of the eigenvectors with respect to the valence basis. Coefficients with a norm smaller than `eigthreshold` will be skipped. This threshold can be set with the option `eigthreshold x`, default `x=.01`.

EIGGAMMA

Prints in the output the eigenvectors in the gamma point.

MEMORY

Print the estimate of the memory usage taken into account to determine the block size and the number of K points treated together.

MULLIKENOVERLAPPOPULATIONS

Prints overlap population of all basis functions.

ORBITALLABELS

Prints the labels of the orbitals. If you are interested in the labels of an old calculation and you don't want to repeat it completely, you can add the option `stopAfter gemtry` to your input file.

ORBPOP

Prints the Mulliken population per orbital, for all eigenstates. This is the most detailed population analysis that you can get, one for all k-points and for each band. Populations below a certain threshold are ignored. This threshold can be set with the option `popthreshold x`. By default `x=.01`.

BLCKAT

Print the information about the distance effects used in the numerical integrals.

RELATIVISTIC

Includes a relativistic correction in the Hamiltonian. This key replaces the keys ZORA and SRZORA of the previous versions of ADF-BAND.

| RELATIVISTIC {level} {formalism} {potential}

Level

May be `NONE` (no relativistic effects), `Scalar` (default) or `SpinOrbit`.

Formalism

Only `ZORA` is supported (no Pauli)

Potential

Only `frozen` is supported (no full)

RESPONSE

Perform a time-dependent DFT calculation to obtain real and imaginary parts of frequency-dependent dielectric function.

```
RESPONSE
  nfreq  5
  strtfr 0.0
  endfr  0.01
  cnvi   0.001
  cnvj   0.001
  ebndt1 0.001
  ifx    0
  isz    0
  iyxc   0
END
```

Omitting the specific options in the `RESPONSE` block will cause default setting to be used during the calculation, as given above.

`nfreq` the number of frequencies in a.u. for which a TDDFT calculation is performed when calculating the dielectric function $\epsilon_e(\omega)$ of a system (default=5).

`strtfr` is the start frequency in a.u. of the frequency range over which the dielectric function is calculated (default=0d0).

`endfr` is the end frequency in a.u. of the frequency range over which the dielectric function is calculated (default=1d-2).

`cnvi` the first convergence criterium for the change in the fitcoefficients for the fitfunctions, when fitting the density (default=1d-3).

`cnvj` the second convergence criterium for the change in the fitcoefficients for the fitfunctions, when fitting the density (default=1d-3).

`ebndt1` the energy band tolerance, for determination which routines to use for calculating the numerical integration weights, when the energy band posses no or to less dispersion (default=1d-3).

`ifxc` integer indicating which fxc kernel is used (default=0).

0 = Adiabatic Local Density Approximation (ALDA) (Can. J. Phys. 58, 1200 (1985)).

1 = Gross-Kohn, frequency dependent fxc kernel (PRL 55, 2850 (1985), 57, 923 (1986)),

2 = van Leeuwen-Baerends (LB94) (PRA 49, 2421 (1994))

`isz` integer indicating whether or not scalar zeroth order relativistic effects are included in the TDDFT calculation (default=0).

0 = relativistic effects are not included,

1 = relativistic effects are included.

iyxc integer for printing *ycx*-tensor (default=0) (JCP 115, 1995 (2001)).

0 = not printed,

1 = printed

RESTART

Tells the program that it should restart with the restart file.

```
RESTART filename {&
  option
END}
```

where *filename* is the name of the restart file and *option* is the program part to do a restart for (SCF, DOS, OrbitalPlot). In its simple form the calculation will do a restart of the SCF. The option for SCF restart is currently disabled, as is OrbitalPlot. Advantage of a restart is the possibility to skip certain (time consuming) program parts.

SCF

Contains the same data as the ADF-MOL key with the same name (except for DIIS procedure parameters). See also second list.

Mixing

Initial 'damping' parameter in the SCF procedure, for the iterative update of the potential: new potential = old potential + mix (computed potential-old potential). Default=0.075.

Note: the program automatically adapts *Mixing* during the SCF iterations, in an attempt to find the optimal mixing value.

Iterations

The maximum number of SCF iterations to be performed. If zero (default value) termination of the SCF procedure will depend only on other aspects (convergence, time-out, insufficient progress towards convergence, ...).

STOPAFTER

Specifies that the program is stopped after execution of a specified program-part (subroutine). The specified name should be one of a pre-defined list. The most relevant ones are *gemtry* (all geometrical aspects are checked, symmetry analysis is carried out, and numbers of (symmetry-unique) integration points in real space and in k-space are determined; this part takes only little cpu-time) and *atomic* (in addition to the geometry-part all radial parts of basis- and fit functions are generated, and the spherically symmetric atoms are computed and inserted in the crystal; the initial charge density is defined and integrated (check on integration-precision), and the electrostatic interaction is computed between the unrelaxed free atoms).

TAILS

Ignore function tails. By default no tails are ignored. Both CPU time and disk space can be saved by using the TAILS option. This option is most effective when combined with the Confine suboption, as explained in the Examples document.

```
TAILS {bas crbas} (core crcore)
```

One real argument for keys *basis* and *core*, which should be a small value; default zero. The *core* criterion defaults to the *basis* criterion (if set). In our experience, a safe value is 1e-5. The tail criterion specifies that tails of exponentially decaying (basis) functions are ignored, in the construction of Bloch functions, beyond the point where the remaining part of the function tail (radially) integrates to less than the criterion, relative to the integral of the function from zero to infinity. This key must be used together with the *Basis* subkey

`NonOrthogonalSCFBasis`, which is recommended anyway. This option has some refinements. The advised settings for optimal performance without significant loss of accuracy is

```
Tails confine=1e-2 bas=1e-5 coulomb
```

As you see there are two new elements ('confine' and 'coulomb'). The first (confine =) specifies that all basis functions are optimized for the tails option, by multiplying the tail of the function with a rapidly decaying function. This step affects the shape of all functions outside the radius where the relative norm of the function is smaller than 1e-2. The effects on the shape of the functions are typically very small. The second new entry (Coulomb) has the effect that the criterion becomes more strict for tight functions. For safer (but slower) calculations, specify smaller values for confine and bas, such as confine=1e-3 bas=1e-6.

The `TAILS` keyword works most effectively when combined with the `confinement` keyword, described elsewhere in this document. The confine key described here should not be confused with the `confinement` option for each atom type. The confine option here affects functions in the region where they become very small, independent of the distance to the nucleus. The `confinement` option introduces a soft cut-off for all functions of a particular atom type, at a specific distance from the nucleus.

TITLE

Compulsory key. Specifies the title of this run. The title is used for identification of the result files.

UNITS

Units of length and angle Geometric lengths and angles are in units defined by this key.

```
UNITS
  {LENGTH {Angstrom | Bohr}}
  {ANGLE {Degree | Radian}}
END
```

Angstrom and *Bohr*, respectively *Degree* and *Radian*, are recognized strings. Each of the subkeys is optional, as is the key UNITS itself. Defaults: Angstrom for lengths, and Degree for angles.

The position of this key in the input is not important. It always applies to **all input**.

To avoid mistakes one should place UNITS as early as possible in input (if at all).

UNRESTRICTED

In the previous versions of the ADF-BAND program, the number of independent spins could be specified by the key `spin`. Other keys that are related to this key are `nspinstartat`, `nspinrefat`. These keys have now all been replaced by this single `Unrestricted` key. If this key occurs (no additional data needed), a spin-*unrestricted* calculation will be carried out.

```
UNRESTRICTED {Startup | Reference | OnlyReference}
```

Reference

Default the formation energy is calculated with respect to the spherically symmetric spin-*restricted* atoms. If you want to do an *unrestricted* calculation for the atoms, you may include this keyword as argument to key `Unrestricted`.

Startup

By default the program uses as a first guess for the density the sum of the spherically symmetric spin-*restricted* atoms. In case you do a spin-*unrestricted* calculation, you may try to use the sum of the *unrestricted* atoms as start-up density. Supplying `Startup` as argument of key `Unrestricted` will give you as start-up density the sum of *unrestricted* atoms with their net spin 'up'. In combination with a frozen core this option can lead to a locally negative valence density, in which case the program will stop and tell you

not to use this option.

OnlyReference

If you want a restricted calculation with as reference the unrestricted atoms, specify this argument to key `Unrestricted`.

Defaults and special cases

A calculation is default *restricted*.

General remarks

Be aware that cpu-time and disk space requirements are approximately doubled!

XC

Specifies the exchange-correlation.

In release 1999.x and before it was a simple key (to specify only the LDA part of the XC functional). With release 2000 it has become a block type key with the same format as in the molecular ADF code: The Density Functional, also called the exchange-and-correlation (XC) functional, consists of an LDA and a GGA part. LDA stands for the Local Density Approximation, which implies that the XC functional in each point in space depends only on the (spin) density in that same point. GGA stands for Generalized Gradient Approximation and is an addition to the LDA part, by including terms that depend on derivatives of the density. For both terms ADF supports a large number of the formulas advocated in the literature.

In principle you may specify different functionals to be used for the *potential*, which determines the self-consistent charge density, and for the *energy* expression that is used to evaluate the (XC part of the) energy of the charge density. To be consistent, one should generally apply the same functional to evaluate the potential and energy respectively. Two reasons, however, may lead one to do otherwise:

1. The evaluation of the GGA part in the *potential* is rather time-consuming. The effect of the GGA term in the potential on the self-consistent charge density is often not very large. From the point of view of computational efficiency it may, therefore, be attractive to solve the SCF equations at the LDA level (i.e. not including GGA terms in the potential), and to apply the full expression, including GGA terms, to the energy evaluation *a posteriori*: post-SCF.
2. A particular XC functional may have only an implementation for the potential, but not for the energy (or vice versa). This is a rather special case, intended primarily for fundamental research of Density Functional Theory, rather than for run-of-the-mill production runs.

The key that controls the Density Functional is `xc`, with sub keys `LDA` and `GGA` (or equivalently: *gradients*) to define the LDA and GGA parts of the functional. Either subkey is optional (need not be used) and may occur twice in the data block: if one wants to specify different functionals for potential and energy evaluations respectively, see above.

```

XC
  {LDA {Apply} LDA {Stoll}}
  {GGA {Apply} GGA}
END

```

Apply

States whether the functional defined on the pertaining line will be used self-consistently (in the SCF-potential), or only post-SCF, i.e. to evaluate the XC energy corresponding to the charge density. The value of `apply` must be `SCF`, `Energy` or `Always`.

A value `postSCF` will also be accepted and is equivalent to `Energy`.

A value `Potential` will also be accepted and is equivalent to `SCF`.

For each record separately the default (if no `Apply` value is given in that record) is `SCF`.

For each of the two terms (LDA, GGA) in the functional: if no record with `Energy` specification is found in the data block, the evaluation of the XC energy will use the same functional as is applied for the potential. If a GGA (Generalized Gradient Approximation) is used in the density functional, applying it is by default

suppressed in the early stages of the SCF procedure, to save CPU time (the evaluation of GGA potentials may be rather time consuming). Using `Always` cancels this feature so that any GGA is evaluated at every cycle of the SCF, including the initial ones.

LDA

Defines the LDA part of the XC functional and can be any of the following:

`xonly`: The pure-exchange electron gas formula. Technically this is identical to the Xalpha form (see next) with a value 2/3 for the X-alpha parameter.

`xalpha`: the scaled (parameterized) exchange-only formula. When this option is used you may (optionally) specify the X-alpha *parameter* by typing a numerical value after the string Xalpha (separated by a blank). If omitted this parameter takes the default value 0.7

`vwn`: the parameterization of electron gas data given by Vosko, Wilk and Nusair (ref [1], formula version V). Among the available LDA options this is the more advanced one, including correlation effects to a fair extent.

Stoll

For the VWN or GL variety of the LDA form you may include Stoll's correction [2] by typing `stoll` on the same line, after the main LDA specification. You must not use Stoll's correction in combination with the Xonly or the Xalpha form for the Local Density functional.

GGA

Specifies the GGA part of the XC Functional, in earlier times often called the "non-local" correction to the LDA part of the density functional. It uses derivatives (gradients) of the charge density. Separate choices can be made for the GGA exchange correction and the GGA correlation correction respectively. Both specifications must be typed (if at all) on the same line, after the GGA subkey.

For the exchange part the options are:

`Becke`: the gradient correction proposed in 1988 by Becke [3].

`PW86x`: the correction advocated in 1986 by Perdew-Wang [4].

`PW91x`: the exchange correction proposed in 1991 by Perdew-Wang [5].

For the correlation part the options are:

`Perdew`: the correlation term presented in 1986 by Perdew [6].

`PW91c`: the correlation correction of Perdew-Wang (1991), see [5].

`LYP`: the Lee-Yang-Parr 1988 correlation correction, [7-9]

Some GGA options define the exchange and correlation parts in one stroke. These are:

`PW91`: this is equivalent to `pw91x + pw91c` together.

`BLYP`: this is equivalent to Becke (exchange) + LYP (correlation).

`LB94`: this refers to the XC functional of Van Leeuwen and Baerends [10]. There are no separate entries for the Exchange and Correlation parts respectively of LB94

The XC energies of exchange potential EV93x (due to Engel and Vosko in 1993) and various forms of the Perdew, Burke, Ernzerhof functional (correlation and three forms of exchange) are printed.

The string GGA must contain not more than one of the exchange options and not more than one of the correlation options. If options are applied for both they must be separated by a blank or a comma.

Defaults and special cases

- If the XC key is not used, the program will apply only the Local Density Approximation (no GGA terms). The chosen LDA form is then VWN.
- If only a GGA part is specified, omitting the *LDA* sub key, the LDA part defaults to VWN, except when the LYP correlation correction is used: in that case the LDA default is Xonly: pure exchange.
- The reason for this is that the LYP formulas assume the pure-exchange LDA form, while for instance the Perdew-86 correlation correction is a correction to a *correlated* LDA form. The precise form of this correlated LDA form assumed in the Perdew-86 correlation correction is not available as an option in ADF but the vwn formulas are fairly close to it.
- Be aware that typing only the sub key *LDA*, without an argument, will activate the VWN form (also if LYP is specified in the GGA part).
- The LB94 functional has only a SCF (=Potential) implementation, but no Energy counterpart. Therefore, LB94 must not be used together with the Energy specification for Apply. If LB94 is used for the Potential (SCF), the GGA energy expression defaults to Becke (exchange part) + Perdew (correlation). This can be overruled by selecting another choice in the 'GGA Energy ...' specification.
- The LB94 form is a density functional specifically devised to get the correct asymptotic behavior. This yields much better energies for the highest occupied molecular orbital (homo). Energies for lower lying orbitals (sub-valence) should improve as well. The energy expression underlying the LB94 functional is very inaccurate. This does not affect the response properties but it does imply that the energy and its derivatives (gradients) should not be used because lb94-optimized geometries will be wrong, see for instance [11].

General remarks

- The phrase non-local in the discussion of density functionals does not mean that non-local potentials are involved. The potentials are perfectly local, but when you go beyond LDA and include gradient corrections, the value of the density functional potential in a point *r* is evaluated not only from the local value of the charge density, but also from the gradient of the charge density.
- The Stoll formula is considered to be a *correlation* correction to the *Local* Density Approximation. It is conceptually not correct to use the Stoll correction *and* apply non-local gradient (GGA) corrections to the correlation. It is the user's responsibility, in general and also here, to avoid using options that are not solidly justified theoretically.
- It is questionable to apply gradient corrections to the *correlation*, while not doing so at the same time for the exchange. Therefore the program will check this and stop with an error message. This check can be overruled with the key allow.
- The issue of the 'best' density functional is a subject of extensive and widespread research. It is generally recognized that applying gradient corrections to the simplest Local Density Approximation usually gives better results for comparison with experimental data, especially as regards bond energies and the spectra computed from one-electron energies.
- The incorporation of gradient corrections during the SCF significantly increases the computing effort. In this respect it makes no difference which specific GGA formula is applied. BAND always prints the energies for all functionals at the end of the SCF, using the converged LDA density. Including the GGA during the SCF starts halfway the SCF unless the key Always is used (see key xc).

Second List

[ALLOW](#) • [BASIS](#) • [CONFINED](#) • [CONFINEMENT](#) • [CONVERGENCE](#) • [CPVECTOR](#) • [DEBUG](#) • [DEPENDENCY](#) • [DIIS](#) • [DIRIS](#) • [EIGTHRESHOLD](#) • [FERMI](#) • [FILELIMIT](#) • [INTEGRATION](#) • [IOVECTOR](#) • [KGRP0](#) • [KGRPX](#) • [KSPACE](#) • [NUELSTAT](#) • [NVELSTAT](#) • [PRINT](#) • [OCCUPATIONS](#) • [POPTHRESHOLD](#) • [POTENTIALNOISE](#) • [SCF](#) • [SCREENING](#) • [WORKSPACE](#)

ALLOW

debugging feature to let the program continue even when intermediate results seem to be wrong or very inaccurate. Currently there are only few places in the program where this key is used. Argument to the key is what should be allowed (e.g. BadIntegration).

BASIS (block-type)

Block key containing options and thresholds related to the basis set.

CHECKBASCOROVL

Check the dependency of the valence basis on the frozen core orbitals, by analysis of the core-valence overlap matrix. Since this takes some cpu time, it is by default off.

LOWDIN

Applies only in fragments calculations. By default the start-up density is taken as sum-of-fragments. This key specifies that the density is constructed from the fragment orbitals after these have been mutually orthonormalized (Pauli principle). It's only related to SCF convergence considerations. Depending on the system it may or may not improve the required number of SCF cycles.

NOCOREDISEPERSION

By default it is assumed that the core states display some small but non-negligible dispersion. Using this key counteracts this assumption. The core Bloch functions are then computed only in the G-point and assumed to be identical for all k-points in the Brillouin Zone.

NONORTHOGONALSCFBASIS

By default the Bloch sums of elementary one-center basis functions are transformed to an orthonormal basis (by numerical integration). Using this key prevents that; in the SCF procedure diagonalization of the Fock matrix is replaced by a general eigenvalue solver that takes the non-trivial overlap matrix into account.

This relatively new option is more efficient than the default and is warmly recommended. It is likely that it will become the default option in future BAND versions.

TRANSEPS

The real argument (default zero) sets a threshold for setting elements to zero in transformation matrices (typically, for transforming the bloch basis to an orthonormal one, and similar transformations) whenever the absolute value is below the threshold.

CONFINED (block-type): global version (not recommended!)

Instead of this global confine option, we recommend the smooth confinement of each atom type, making use of a Fermi-Dirac function. This preferred option is documented below and should lead to numerically more stable results.

A confinement approach has been implemented in BAND to squeeze the radial atomic functions a bit faster to zero. This can be achieved using three different methods: multiplication by a confining function (method exponential), by replacing the atomic function (method polynomial) or by scaling the atomic function (method scaling).

```

CONFINEMENT
  { Method { exponential | polynomial | scaling } }
  { Rmatch rmin }
  { Rcut rmax }
  { Order norder }
End

```

The confinement is applied between *rmin* and *rmax*, though due to renormalization, the entire function will change (also in the inner region). Before *rmin* the function will not be adapted apart from the renormalization, after *rmax* the function becomes zero. Obviously, confinement of functions may have some impact on the numerical outcomes. The value *norder* is used for the scaling technique. Currently only works for non-relativistic calculations.

CONFINEMENT (block-type): subkey of atomtype

The global confinement option leads to discontinuous higher order derivatives in the basis function, leading to numerical instability. It is better to use smooth confinement. Smooth confinement means that the radial part of a function is multiplied with a Fermi-Dirac (FD) function. We have implemented smooth confinement at the atom type level. In a slab calculation this allows one to use different settings for surface atoms than for those in inner layers. You can specify the range and the decay speed of the FD function like

```

Confinement
  Radius 7
  Delta 0.7
SubEnd

```

If the decay 'Delta' is not specified it defaults to $0.1 \cdot \text{Radius}$. Relativistic effects are treated correctly. If the confinement option is used in combination with the TAILS option, the calculation may run significantly faster. The `confinement` option can also be useful without the tails option in cases where near linear dependency reduces the numerical reliability of the results (cf. `dependency` keyword). This typically occurs for highly coordinated systems with large basis sets containing diffuse functions. For such cases, the `confinement` option (possibly in combination with the `dependency` keyword) reduces the numerical problems.

CONVERGENCE (block-type)

All options and parameters related to the convergence behavior of the SCF are defined in this block key. Also the finite temperature distribution is part of this.

BOLTZMANN

Number of points by which the finite temperature Fermi-Dirac distribution of electronic occupations is approximated. Advise: don't use.

LESSDEGENERATE

If smoothing of occupations over nearly degenerate orbitals is applied (see the key DEGENERATE), then, if this key is set in the input file, the program will limit the smoothing energy range to $1e-4$ a.u. as soon as the SCF has converged 'halfway', i.e. when the SCF error has decreased to the square root of its convergence criterion.

NODEGENERATE

This key prevents any internal automatic setting of the key DEGENERATE, see that key's description.

TEMPERATURE

Defines the distribution of occupations around the Fermi level. By default ($T=10$), the effect is that of zero temperature. In fact this key is a preliminary to a future full implementation of finite temperature effects; currently it has no sensible application. See, however, the ALLBANDS key.

CPVECTOR

The code is vectorized and this key can be used to set the vector length. Default depends on the machine and should be set at the installation of the program.

DEBUG

To quickly implement a new key. DEBUG takes as argument a key name. The programmer can check for this in the same way as PRINT or ALLOW keys.

DEBUG LNOSPN

Applies only in a spin-unrestricted calculation. The program assumes that energy bands are constituted of the results at the discrete k-points that correspond in energy-ordering: the first band is made up of all lowest eigenvalues across the BZ, the second band of the second lowest values, et cetera. This procedure is, by default, carried out independently for both spins. Using the key 'mixes' spin-alpha and spin-beta orbitals and allows spin-mixed bands, so to speak. This affects the calculation of occupation numbers in case of partially filled bands. It is primarily a testing and debugging tool.

DEPENDENCY

Criterion for dependency of the basis and fit set.

```
| DEPENDENCY {basis tolbas} {core tolcor} {fit tolfit} {corevalence tolovl}
```

basis

Smallest eigenvalue of the overlap matrix of normalized bloch functions. Default 1e-5. See also the discussion in 'Recommendations & Problems' about basis set dependency.

core

The program verifies that the frozen core approximation is reasonable, by checking the smallest value of the overlap matrix of the core (bloch) orbitals against this criterion. Default: 0.98

fit

Criterion for dependency of the total set of fit functions. The value monitored is the smallest eigenvalue of the overlap matrix of normalized bloch sums of symmetrized fit functions. Default=1e-6.

corevalence

Criterion for dependency of the core functions on the valence basis. The maximum overlap between any two normalized functions in the two respective function spaces should not exceed $1.0 - \text{coreval}(\dots)$. Default=1e-5.

DIIS (block-type)

The diis procedure to obtain the SCF solution in the crystal depends on several parameters. Default values can be overruled with this key-block. Each option must be specified, it at all, on a separate record in the data block:

```
| DIIS
|   option1  value1
```

```

    option2  value2<
    ...
END

```

Recognized options are:

condition: the condition number of the DIIS matrix, the largest eigenvalue divided by the smallest, must not exceed this value; default 1e6

large: when the largest coefficient in the DIIS expansion exceeds this value, damping is applied; default 20

chuge: when the largest DIIS coefficient exceeds this value, the oldest DIIS vector is removed and the procedure re-applied. Default 50

dimix: mixing parameter when damping is used, rather than the DIIS procedure. By default off (dimix=1.0): result is taken as $\text{dimix} \times \text{value} + (1 - \text{dimix}) \times \text{previous}$

nvctrx: maximum number of DIIS expansion vectors. Default 20

ncycledamp: number of initial iterations where damping is applied, before any DIIS is considered. Default 5

potential: (no argument): presence of this string means that the DIIS method will be applied to the *potential*, rather than to the valence density (default). It excludes the competing option deformationdensity.

deformationdensity: (no argument): presence of this string means that the DIIS method will be applied to the *deformation density*, rather than to the valence density (default). This option excludes the potential option, see previous.

print: turns on a print switch to report details of the used DIIS procedure. Default off.

special: (no argument) will let the program try to optimize the mixing parameter (dimix) and adjust it when difficulties occur. It is not certain that this may not make things worse!

comstr: its argument is only a string to be printed as label to output, if any, of the DIIS parameter.

DIRIS (block-type)

Completely similar to the DIIS key, except that this one applies to the DIIS procedure used in the DIRAC subprogram, for numerical single atom calculations.

EIGTHRESHOLD

Components smaller (absolute value) than this parameter (default 1e-2) are not printed in the output of the DOS section, where the breakdown of crystal orbitals in the primitive basis is output.

FERMI (block type)

This key sets technical parameter used in the search for the Fermi energy, which is carried out at each cycle of the SCF procedure. All applicable options must be specified, if at all, in separate records in the data block.

```

FERMI
    option1  value1
    option2  value2
    (etc)
END

```

Recognized options:

maxtry: maximum number of attempts to locate the Fermi energy accurately. Default 50. The procedure is

iterative in nature, narrowing the energy band in which the Fermi energy must lie, between an upper and a lower bound. If the procedure has not sufficiently converged within maxtry iterations, the program takes a reasonable value and constructs the charge density by interpolation between the functions corresponding to the last used upper and lower bounds for the Fermi energy

delta: converge criterion: upper and lower bounds for the Fermi energy and the corresponding integrated charge volumes must be equal within delta. Default 1e-4

eps: after convergence of the Fermi energy search procedure, a final estimate is defined by interpolation and the corresponding integrated charge volume is tested. It should be exact, to machine precision. Tested is that it deviates not more than eps. Default 1e-10

FILELIMIT

maximum amount of data (in bytes) on one logical file. Default depends on the machine and should be set at the installation of the program. Advise: Choose as large as possible, to keep the number of files limited. The automatic algorithm that cuts the files to pieces does not work flawlessly. Using a high file limit effectively disables the use of this mechanism. Be aware however of the maximum integer value on your machine!

INTEGRATION (block-type)

parameter-specifications for the generation of numerical integration points and weights. Most data records must be of the form 'parameter value'. The most important parameter is `accint`, which is defaulted to the value of key `Accuracy`. Unless one is very familiar with the details of the numerical integration package, we strongly recommend not to use the `INTEGRATION`-key, and to specify only `Accuracy`. More information can be found in the literature. A key that is specific to BAND (cannot be used in ADF) is the key `Pirpt3`. This invokes a slightly different generator for the numerical integration grid. Like the standard method, it is based on an atomic cellular partitioning of space, but it differs in the treatment of the truncated pyramids, by more strictly monitoring test functions. It generates more points (which means: increased CPU times and disk storage), and in some cases it yields more accurate results. General advise is hard to give.

IOVECTOR

I/O actions to and from files is segmented in blocks of `iovector`. Default depends on the machine and should be set at the installation of the program.

KGRP0

treatment of the k-points (integration over the Brillouin Zone) in the preparation phase (construction of Bloch basis functions, computation of overlap matrix, and so on, in each k-point) is in blocks of `kgrp0` points at the same time. Note: last character of this key is a zero (not the letter "o").

During the preparation phase increasing `kgrp0` may reduce the CPU time. This depends also on available workspace). However, larger `kgrp0` values result in more files being open at the same time, and more data being stored on disc during this stage of the program. By default the program tries to optimize `kgrp0` only with respect to the expected effect on cpu-time.

If the key is used, the actual value of `kgrp0` may differ slightly from the input-value: from the input-value the program computes first the number of *blocks* of k-points; then `kgrp0` is re-computed by distributing the total number of k-points equally over the blocks. To restrict the size of the blocks via input it is most convenient to use the key `kgrpX`, rather than `kgrp0`.

KGRPX

is an absolute upper bound on `kgrp0` as computed by the program.

KSPACE (general-type)

Various settings of the k-space integration can be supplied here. The simple form of this key is the accuracy of the integration and can be specified within the block key as `Kinteg` (see also the first list).

Hybrid

invokes the *hybrid* quadratic (rather than *fully*) quadratic integration method over the BZ. It is meaningful only for 2D Brillouin Zones that would otherwise use a fully quadratic procedure (odd-valued k-space integration parameter). In all other cases the key is ignored.

KInteg

See key κ space in the first list. This defines the accuracy of the integration of the reciprocal space.

KMesh

secondary parameter for integration over the Brillouin Zone: some aspects in the quadratic method may be carried out in fact by using a fine-grid linear-tetrahedron method (: hybrid approach). *kmesh* defines this linear-method mesh. Default=2 (invariably found to be adequate).

LinearKspace

Integration over the BZ is carried out with the analytic quadratic tetrahedron method. If the input κ SPACE key is set to an *even* number, this approach is not used and the *linear* tetrahedron method is used (usually far inferior). To invoke the linear method also for *odd* values of κ SPACE, insert the *LinearKspace* key (no argument) in the input file.

NonSymSampling

The BZ sampling grid is generated in a summation over simplices that build the irreducible wedge of the BZ, considering only the geometric symmetry of the BZ itself. Any additional k-points required, when the atomic positions in the unit cell imply symmetry lowering, are added by symmetry operations. In pre-98 releases of BAND the BZ grid was generated in the symmetry unique part of the BZ considering the real space symmetry (which could, therefore, be a lower symmetry) directly. The difference between the two methods is in particular relevant when doing comparison calculations where the systems differ only in the atomic positions implying different space group symmetries. One would then want to use the same BZ grid. This is the case since release 98, but was not (by default) so in earlier versions. The original approach can be selected by using this key..

SUPPRESS

Suppresses integration in k-space in one or more directions. May be used for instance if the 3D Brillouin Zone is very extended in one or two dimensions and of 'normal' size in the other dimension(s). We plan to remove this key in the future. If it is given in input, with an integer value, integration will be suppressed in the indicated number of dimensions.

NUELSTAT

Electrostatic interaction integrals between spherical atomic densities are computed by numerical integration over an elliptic grid. *Nuelstat* is the outward (parabolic) coordinate number of integration points.

Default: 50

NVELSTAT

Electrostatic interaction integrals between spherical atomic densities are computed by numerical integration over an elliptic grid. *Nvelstat* is the angular (elliptic) coordinate number of integration points.

Default: 80

PRINT

One or more strings (separated by blanks or comma's) from a pre-defined set may be typed after the key. The following names replace the *IRPNT*[*IPRSE*] keys of the previous versions of BAND

PrepNone, PrepMore, PrepDetail replace IPRNTP 0, 2, 5

IntNone, IntMore, IntDetail replace IPRNTI 0, 2, 5

FrmNone, FrmMore, FrmDetail replace IPRNTR 0, 2, 5

SCFNone, SCFMore, SCFDetail replace IPRNTS 0, 2, 5

EigNone, EigMore, EigDetail replace IPRNTE 0, 2, 5

OCCUPATIONS (block-type)

Allows to input specific occupations numbers. Applies only for calculations that use only one k-point (i.e. pseudo-molecule calculations).

```
OCCUPATIONS
  irrepro occupations_alpha {// occupations_beta}
  ...
End
```

the *irrepro* must be 1, unless symmetry is used (an unsupported option, currently). *occupations_beta*, and the separating double slash (*//*) must not be used in a spin-restricted calculation. *occupations_alpha/beta* is a sequence of values assigned to the states ('bands') in energy ordering. This allows you, for instance, to specify an empty state below occupied ones.

POPTHRESHOLD

Threshold for printing Mulliken population terms. Default 1e-2

POTENTIALNOISE

The initial potential for the SCF procedure is constructed from a sum-of-atoms density. Added to this is some small noise in the numerical values of the potential in the points of the integration grid. The purpose of the noise is to help the program break the initial symmetry, if that would lower the energy, by effectively inducing small differences between (initially) degenerate orbitals. The noise in the potential is randomly generated between zero and an upper limit, which is 1e-4 a.u. by default. The key, which must have a numerical argument, adjusts this upper limit. This can be used therefore to suppress the noise by choosing zero, or to increase it by specifying some large number.

SCF

Contains the same data as the ADF-MOL key with the same name (except for DIIS procedure parameters).

EigenStates

The program knows two alternative ways to evaluate the charge density iteratively in the SCF procedure: from the P-matrix, and directly from the squared occupied eigenstates. By default the program actually uses both at least one time and tries to take the most efficient. *Eigenstates* turns off this comparison and lets the program stick to one method (from the eigenstates).

Pmatrix

Evaluate the charge density from the P-matrix. See also the subkey *Eigenstates*.

Rate

Minimum rate of convergence for the SCF procedure. If progress is too slow the program will take measures (such as smearing out occupations around the Fermi level, see subkey *degenerate* of key *convergence*) or, if everything seems to fail, it will stop. Default=0.99

VSPLIT

To disturb degeneracy of alpha and beta spin MOs the value of this key is added to the beta spin potential at the startup. Default is 5E-2.

SCREENING (block)

Parameters that influence the screening and tails of basis functions. Recognized options are

CUTOFF

Criterion for negligibility of tails in the construction of Bloch sums. Default depends on *Accuracy*.

DMADEL

One of the parameters that define the screening of Coulomb-potentials in lattice sums. Depends by default on *Accuracy*, *rmadel*, and *rce1x*. One should consult the literature for more information.

RCELX

Max. distance of lattice site from which tails of atomic functions will be taken into account for the bloch sums. Default depends on *Accuracy*.

RMADEL

One of the parameters that define screening of the Coulomb potentials in lattice summations. Depends by default on *Accuracy*, *dmadel*, *rce1x*. One should consult the literature for more information.

NODIRECTIONALSCREENING

Real space lattice sums of slowly (or non-) convergent terms, such as the Coulomb potential, are computed by a screening technique. In previous releases, the screening was applied to all (long-range) Coulomb expressions. Starting from BAND98 screening is only applied in the periodicity directions. This key restores the original situation: screening in all directions.

SKIP

followed by any number of strings (separated by blanks or commas) tells the program to skip certain parts. Should only be used by those who know what they're doing. Recognized are certain pre-defined strings. Useful argument may be *eigenvalues* (to suppress printing the eigenvalues at the (first and last) SCF cycles).

WORKSPACE (block-type)

Options for the workspace manager. Do not use, but use settings file.

INTEGERMEMBLOCK

smallest block of memory to allocate to store integers in (in Megabytes).

LOGICALMEMBLOCK

smallest block of memory to allocate to store logicals in (in Megabytes)

REALMEMBLOCK

smallest block of memory to allocate to store reals in (in Megabytes).

STRINGMEMBLOCK

smallest block of memory to allocate to store strings in (in Megabytes).

Third List

ACCINT • ALLBANDS • ALWAYSNLPOI • ANGSTROM • BASISDEPEND • BASISFUNCTIONS • BOLTZMAN • CONTINUE • CONVERGENCE • COREDEPEND • COREFUNCTIONS • COREVALENCEDEPEND • CUTOFF • CYCLES • DEBUG • DEBUGWORKSPACEMANAGER • DEGENERATE • DFL • DFRMAX • DFRMIN • DFRNUC • DFVALENC • DIRAC • DMADEL • DOTEST • EDEGEN • ESMEAR • EXECUTE • FILELENGTH • FITDEPEND • FITFUNCTIONS • FRG • GRADBAS • HYBRID • IPRNTE • INTEGERMEMBLOCK • KMESH • LESSDEGENERATESMOOTHING • LINEARKSPACE • LNOSPN • LOGICALMEMBLOCK • LOWDIN • MIX • NLXCPOI • NOCOREDISP • NODEGENERACY • NONORTHOGONALSCFBASIS • NSPINSTARTAT • NSPINREFAT • OLDKPOINTS • ORBPLOI • ORBLABELS • PFDIRECTORY • PIRPT3 • PREPAREFRAGMENT • REALMEMBLOCK • RESTARTDIRECTORY • RESTARTDOS • RESTARTSCF • RESTARTFILE • RESULTDIRECTORY • RESULTFILE • RHOCHOICE • RMADEL • SCFLINTHRESH • SCFRATE • SHARP • SPIN • SRZORA • STRINGMEMBLOCK • SUPPRESS • TAILCR • TEST • TEMPERATURE • TRAEPS • ZORA

ACCINT

Obsolete. Replaced by Accuracy.

ALLBANDS

Moved to blockkey Convergence.

ALWAYSNLPOI

Obsolete. Replaced by xc.

ANGSTROM

Obsolete. Replaced by Units.

BASISDEPEND

Obsolete. Replaced by Dependency.

BASISFUNCTIONS

Moved to blockkey AtomType.

BOLTZMAN

Obsolete. Use subkey Boltzmann of key Convergence.

CONTINUE

Obsolete. Replaced by Allow.

CONVERGENCE

Changed. Now a block key. Functionality replaced by key Criterion within this block key.

COREDEPEND

Obsolete. Replaced by Dependency.

COREFUNCTIONS

Moved to blockkey `AtomType`. Currently disabled.

COREVALENCEDEPEND

Obsolete. Replaced by `Dependency`.

CUTOFF

Moved to blockkey `Screening`.

CYCLES

Obsolete. Use subkey `Iterations` of key `SCF`.

DEBUG

Changed. Now takes as argument a debug key option. Not intended for normal calculations.

DEBUGWORKSPACEMANAGER

Obsolete. Turn on the debug mode for the workspace manager by specifying `Debug WorkspaceManager`.
Note extra space!

DEGENERATE

Moved to blockkey `Convergence`.

DFLF

Obsolete. Value can only be set inside the `Dirac` subkey of `AtomType` (`NumericalFit`).

DFRMAX

Obsolete. Value can only be set inside the `Dirac` subkey of `AtomType` (`Rmax`).

DFRMIN

Obsolete. Value can only be set inside the `Dirac` subkey of `AtomType` (`Rmin`).

DFRNUC

Obsolete. The `DIRAC` subprogram takes this as the size for the nucleus. Default 0. Note that only one value can be set, which will apply to all atoms. Therefore, this key is not very useful in its current implementation, unless one uses only one type of atom in the calculation. Value can only be set inside the `DIRAC` subkey of `AtomType` (`Rnuc`).

DFVALENC

Obsolete. Value can only be set inside the `Dirac` subkey of `AtomType` (`Valence`).

DIRAC

Moved to blockkey `AtomType`.

DMADEL

Moved to blockkey `Screening`.

DOTEST

Obsolete.

EDEGEN

Obsolete. Use subkey Degenerate of key Convergence.

ESMEAR, ISMEAR, NLEVEL, NSMEAR

Obsolete. Smearing is disabled.

EXECUTE

Obsolete. Replaced by StopAfter.

FILELENGTH

Obsolete. Replaced by FileLimit.

FITDEPEND

Obsolete. Replaced by Dependency.

FITFUNCTIONS

Moved to blockkey AtomType.

FRG

Obsolete. Replaced by Fragments.

GRADBAS

Obsolete. Use subkey Gradients of key Basis.

HYBRID

Moved to blockkey Kspace.

IPRNTE, IPRNTI, IPRNTP, IPRNTS, IPRNTR

Obsolete. Use Print keys.

INTEGERMEMBLOCK

Moved to block key WorkspaceManager.

KMESH

Moved to blockkey Kspace.

LESSDEGENERATESMOOTHING

Obsolete. Use subkey LessDegenerate of key Convergence.

LINEARKSPACE

Moved to blockkey Kspace.

LNOSPN

Obsolete. Use Debug LnoSpn.

LOGICALMEMBLOCK

Moved to block key `WorkspaceManager`.

LOWDIN

Moved to block key `Basis`.

MIX

Obsolete. Use subkey `Mixing` of key `SCF`.

NLXCPOPT

Obsolete. This key existed in release 1999.x and before, but has been disabled. Its functionality is replaced by the (new version of the) block type key `XC`.

NOCOREDISEP

Obsolete. Use subkey `NoCoreDispersion` of key `Basis`.

NODEGENERACY

Obsolete. Use subkey `NoDegenerate` of key `Convergence`.

NONORTHOGONALSCFBASIS

Moved to block key `Basis`.

NSPINSTARTAT

Obsolete. Replaced by `Unrestricted`.

NSPINREFAT

Obsolete. Replaced by `Unrestricted`.

OLDKPOINTS

Obsolete. Use subkey `NonSymSampling` in (block) key `Kspace`.

ORBEPLOT

Obsolete. Replaced by `OrbitalPlot`. Currently disabled.

ORBLABELS

Obsolete. Replaced by `OrbitalLabels`.

PFDDIRECTORY

Obsolete. Always use the current executing directory ('.').

PIRPT3

Moved to block key `Integration`.

PREPAREFRAGMENT

Moved to `Basis` blockkey.

REALMEMBLOCK

Moved to block key `WorkspaceManager`.

RESTARTDIRECTORY

Obsolete key. Restart directory is always current directory. File names are w.r.t. current directory or given by an absolute path name.

RESTARTDOS

Obsolete key. Replaced by Restart.

RESTARTSCF

Obsolete key. Replaced by Restart. Currently disabled.

RESTARTFILE

Obsolete key. Replaced by Restart.

RESULTDIRECTORY

Obsolete key. Directory for result file (RUNKF) is always current directory.

RESULTFILE

Obsolete key. Result file is always RUNKF.

RHOCHOICE

(Used with SKIP). Obsolete. Use subkey Eigenstates of key SCF.

RMADEL

Moved to block key Screening.

SCFLINTHRESH

Obsolete. Linear approximation during SCF removed.

SCFRATE

Obsolete. Use subkey Rate of key SCF.

SHARP

Obsolete. Use subkey NoDegenerate of key Convergence.

SPIN

Obsolete. Replaced by Unrestricted.

SRZORA

Obsolete. Replaced by Relativistic.

STRINGMEMBLOCK

Moved to block key WorkspaceManager.

SUPPRESS

Moved to the Kspace block.

TAILCR

Obsolete. Replaced by `Tails`.

TEST

Obsolete.

TEMPERATURE

Moved to block key `Convergence`.

TRAEPS

Obsolete. Use subkey `TransEps` of key `Basis`.

ZORA

Obsolete. Replaced by `Relativistic`.

4.2 Auxiliary Input Features

The program supports the following features to enhance user-friendliness of input:

- You may refer to a key by any *initial* substring, provided no other key has the same initial substring. For instance, if the keys `abcabc` and `abcdef` are known to the program, then `abca` will be understood to denote the key `abcabc`, but using `abc` would be incorrect, and may either result in an error or the program may choose any of the keys that have this initial substring, and not necessarily the one you intended. Initial substrings cannot (yet) be used for the following keys: `define`, `print`, `skip`.
- Arithmetic *expressions* can be used (wherever numbers are required) involving the standard arithmetic operands in Fortran (`+ - * / **`), together with parentheses where necessary or convenient. Blanks are allowed in expressions and ignored, but they are interpreted as separators, i.e. as denoting the end of an expression, whenever the part before the blank can be evaluated as a correct expression. For instance `'3* 4'` will be interpreted as 12, but `'3 *4'` will be interpreted as 3, followed by a character `*`, followed in turn by the number 4. All numbers and results are interpreted and handled as being of type real, but whenever the result is a whole number (allowing for small round-off deviations), it will be recognized as possibly denoting an integer.
- The user may define *variables* and *functions* in the input file, and apply them subsequently in expressions. The input file is read sequentially and variables and functions must be defined before they can be used. Note carefully that replacement of a variable (or function) by its value will occur wherever possible (textually), even if it leads to non-sense input. A frequently occurring mistake is that the user defines a variable `'C'` in his (her) input and then gets his input corrupted because of subsequent isolated `C` characters are replaced by the defined numerical value. *Therefore: avoid single-character variables and function names.* Always check carefully that the identifier you introduce is not 'used' already in the input file.

A few variables and functions are pre-defined:

(variables):

`pi = 3.1415....`

(functions):

`sin, cos, tan, asin, acos, atan, exp, log, sqrt, nint.`

The argument list of a function must be enclosed in parentheses, and the arguments (in case of more than one) must be separated by commas.
 Defining variables and/or functions is done with the block-type key `define`.

Example (part of input):

```
DEFINE
ab = sin(pi/3)
s13 = 14*sqrt(2)
func(x,y,z) = x*ab+y**2-y*z
End
AKEY = FUNC (S13/5, S13/7, SIN(PI/6))
```

Here a function `func` and variables `ab` and `s13` are defined, using the pre-defined functions `sin` and `sqrt`, as well as the pre-defined variable `pi`. These are then applied to assign a value to the (hypothetical) key `AKEY`.

Note 1: the variable `ab` is also used in the definition of `func`; this is allowed because `ab` is defined before `func`.

Note 2: variable- and function *names* must have the same form as keywords, i.e. only certain characters are allowed.

Note 3: in the definition of variables and functions blanks are ignored altogether (in the value part) and will

not be interpreted as possible separators of the expression that defines the variable or function.

- (Single) quotes can be used to designate strings, i.e. (parts of) records which are not to be parsed for expressions, but which are to be taken as they are. The quotes themselves are ignored.
 Double quotes inside a string are understood to denote the single quote character (as part of the string).
- Empty records and starting blanks in records are allowed (and ignored), and can be used to enhance clarity and readability of the input file for human readers by structuring its layout.
- You may exclude certain parts of the input file from being considered by the input-reading routine, which can thus be used for instance to include 'comments' and clarifying remarks for later inspection. This can be done in two ways:
 - A double colon (::) is interpreted by the input routine as denoting the end-of-line, so that the part of the line after it (including the double colon itself) is ignored.
 - The block-type keyword `ignore` starts a block of information that will completely be ignored by the program. The `ignore`-block must end, as usual, with the end-key code.

References

1. Vosko, S.H., L. Wilk, and M. Nusair, *Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis*. Canadian Journal of Physics, 1980, **58**: p. 1200
2. Stoll, H., C.M.E. Pavlidou, and H. Preuss, *On the calculation of correlation energies in the spin-density functional formalism*. Theoretica Chimica Acta, 1978, **49**: p. 143
3. Becke, A.D., Physical Review A, 1988, **38**: p. 3098
4. Perdew, J.P. and Y. Wang, *Accurate and simple density functional for the electronic exchange energy: generalized gradient approximation*. Physical Review B, 1986, **33**(12): p. 8800
5. Perdew, J.P., *et al.*, Physical Review B, 1992, **46**: p. 6671
6. Perdew, J.P., *Density-functional approximation for the correlation energy of the inhomogeneous electron gas*. Physical Review B, 1986, **33**(12): p. 8822
7. Lee, C., W. Yang, and R.G. Parr, *Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density*. Physical Review B, 1988, **37**(2): p. 785
8. Johnson, B.G., P.M.W. Gill, and J.A. Pople, *The performance of a family of density functional methods*. Journal of Chemical Physics, 1993, **98**(7): p. 5612
9. Russo, T.V., R.L. Martin, and P.J. Hay, *Density Functional calculations on first-row transition metals*. Journal of Chemical Physics, 1994, **101**(9): p. 7729
10. van Leeuwen, R. and E.J. Baerends, *Exchange-correlation potential with correct asymptotic behaviour*. Physical Review A, 1994, **49**(4): p. 2421-2431
11. Neumann, R., R.H. Nobes, and N.C. Handy, *Exchange functionals and potentials*. Molecular Physics, 1996, **87**(1): p. 1-36

BAND User's Guide: Index

ACCURACY [1]
ALLBANDS [1]
ALLOW [1]
ANGSTROM [1]
ATOMS [1]
ATOMTYPE [1]
BASIS [1] [2] [3]
BASISFUNCTIONS [1]
BLCKAT [1]
BOLTZMANN [1]
CHECKBASCOROVL [1]
COMMENT [1]
CONFINEMENT [1] [2]
CONVERGENCE [1]
COORDINATES [1]
CPVECTOR [1]
CRITERION [1]
CUTOFF [1]
DEBUG [1]
DEFINE [1]
DEGENERATE [1]
DEPENDENCY [1]
DIIS [1] [2]
DIRAC [1]
DIRIS [1]
DMADEL [1]

DOS [1]
EIGENS [1]
EIGENSTATES [1]
EIGGAMMA [1]
EIGHTHRESHOLD [1]
FERMI [1] [2]
FILELIMIT [1]
FITFUNCTIONS [1]

FORMFACTORS [1]
FRAGMENTLABELS [1]
FRAGMENTS [1]
GGA [1]
GROSSPOPULATIONS [1]
HYBRID [1]
IGNORE [1]
INTEGERMEMBLOCK [1]
INTEGRATION [1]
IOVECTOR [1]
ITERATIONS [1]
KGRP0 [1]
KGRPX [1]
KINTEG [1]
KLABELS [1]
KMESH [1]
KSPACE [1]
LATTICE [1]
LDA [1]
LESSDEGENERATE [1]
LINEARKSPACE [1]
LOGICALMEMBLOCK [1]
LOWDIN [1]
MAXMEMORYUSAGE [1]
MEMORY [1]
MIXING [1]
MULLIKENOVERLAPPOPULATIONS [1]
NATOMSASFRAGMENT [1] [2]
NOCOREDISPERSION [1]
NODEGENERATE [1]
NODIRECTIONALSCREENING [1]
NONORTHOGONALSCFBASIS [1]
NONSYMSAMPLING [1]
NUELSTAT [1]
NVELSTAT [1]

OCCUPATIONS [1]
ORBITALLABELS [1] [2]
ORBITALPLOT [1]
ORBPOP [1]
OVERLAPPOPULATIONS [1]
PLANEWAVES [1]
PMATRIX [1]
POPTHRESHOLD [1]
POTENTIALNOISE [1]
PREPAREFRAGMENT [1]
PRINT [1] [2] [3]
RATE [1]
RCELX [1]
REALMEMBLOCK [1]
RELATIVISTIC [1]
RESPONSE [1] [2]
RESTART [1]
RMADEL [1]
SAVE [1]
SCF [1] [2]
SCREENING [1]
SIMPLEFRAG [1]
STOLL [1]
STOPAFTER [1]
STRINGMEMBLOCK [1]
SUPPRESS [1]
TAILS [1]

TEMPERATURE [1]
TITLE [1] [2]
TRANSEPS [1]
UNITS [1]
UNRESTRICTED [1]
VSPLIT [1]
WORKSPACE [1]
XC [1] [2]