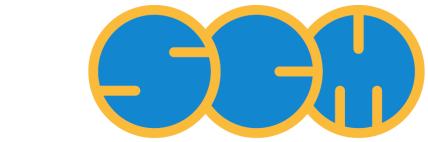

Scientific Computing & Modelling

BAND Manual

ADF Program System
Release 2013

Scientific Computing & Modelling NV
Vrije Universiteit, Theoretical Chemistry
De Boelelaan 1083; 1081 HV Amsterdam; The Netherlands
WWW: www.scm.com
E-mail: support@scm.com

Table of Contents

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 2013.01

Compared to the 2012 release, the 2013 version offers

- Visualization of the Fermi surface
- Calculation of effective mass for electron/hole mobility
- GLLB-SC model potential for band gap
- Finite nucleus model
- Density at nucleus
- Optional Becke grid
- Optional regular grid in k-space
- Various bug fixes and performance improvements

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. Unlike ADF, BAND can use numerical atomic orbitals, so that the core is described very accurately. Because of the numerical orbitals BAND can calculate accurate *total* energies. Furthermore it can handle basis functions for arbitrary *l* values.

Characterization of BAND

Functionality

- Automatic geometry optimization
- 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), GGA (Generalized Gradient Approximation), and meta GGA (with kinetic energy density dependence) formulas
- Time-dependent DFT 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.
- Phonon spectrum.

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
- Local Densities-of-States: LDOS (STM images)
- Form factors (X-ray structures)
- Charge analysis using Voronoi cells (yielding Voronoi Deformation Charges)
- Orbital plots
- Deformation density plots
- Band structure plots along the edges of the Brillouin zone
- 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
- QT-AIM (Quantum Theory of Atoms In Molecules). Atomic charges and critcal points.
- ELF: Electron Localization Function (understand bonding).

Technical

- Linear scaling techniques are used to speed up calculations on large unit cells
- 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.

Changed defaults in the 2010 release

- Fast implementations of overlap (Fock matrix) and dot product (for density) routines
- NMR: use analytical gradient. (See `NMR%Numeric`)
- Numerical frequencies step size changed from 0.01 to 0.001. (See `Frequencies%Step`)

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.

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. Specify the `BasisDefaults` key. With this you can set the quality
   of the basis set to be used, and the size of the frozen core. Given your settings the program will look up an appropriate basis set for each type from the database. This replaces the old style `AtomType`
   key that needed to be specified for each type.
5. Add further keys for features you don't want to be defaulted, for instance
   `Unrestricted` (if you want to do a spin-*un*restricted 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`
7. Comments: everything behind a exclamation mark "!" is treated as comment. If you put it behind a key, leave some space between it, otherwise it will be seen as part of the key.

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

2.2 Structure of the input

General Input Features

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

- 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.

- (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.
- A double colon :: is interpreted as an end of line character, hence the double colon and anything after it is ignored

Title, Comments and Ignore

`Title`

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

`Commment (block-type), and line oriented`

The content of this key is a text that will be copied to the output header, where general program information is also printed. The exclamation mark can also be used to add comment to you input that should not interpreted by BAND. Example

```
Comment ! here are my comments
Description of the calculation
Some more description
End

Lattice ! FCC lattice
0 a a
a 0 a
a a 0
End

...

! Here are my defines

Define
...
End
```

Finally it is possible to let BAND ignore parts of the input

`Ignore (block-type)`

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

Defining variables and functions

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.

`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)

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.

`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: Bohr 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).

Execution flow*

There are a few ways to alter the standard execution flow, which may be useful to developers or expert users.

`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).

`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 `DOS` when that part of the program takes a long time (usually not), or `eigenvalues` (to suppress printing the eigenvalues at the (first and last) SCF cycles).

`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 (possible arguments: `BadIntegration, OCC, CHARGEERROR, DEPENDBASIS`).

2.3 Geometry

Lattice vectors

`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 units. To specify coordinates in Angstrom, use the `Units` key.

Coordinates of atoms in the unit cell

`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.

You can add the following subkeys that override the behavior of the `BasisDefaults` key

`BasisType`

This overrides the setting of the `BasisDefaults%AtomType` key for this atom type.

`Core`

This overrides the setting of the `BasisDefaults%Core` key.

`File`

Specify a full path to a basis set file.

Natural coordinates

`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.

Labels for points in k space

`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.

Selected atoms

`SelectedAtoms`

With this key you can select atoms. This has an effect on a couple of options, like for instance the `Gradients`, Hessian (`FREQUENCIES`), `NMR`, and `EFG` options. In these cases, the properties will only be calculated for the selected atoms, which can make the job faster.

```
SelectedAtoms n1, n2, n...
```

The numbers of the atoms are as on input. If the selection breaks the symmetry the program will stop. Leaving out this key in general selects *all* atoms.

2.4 Model Hamiltonian

The program calculates by default the non relativistic 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).

Density Functional

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.

XC functionals

The density functional approximation is controlled by the `XC` key.

`XC`

Three classes of exchange-correlation functionals are supported: LDA, GGA, and meta-GGA. There is also the option to add an empirical dispersion correction. The only ingredient of the LDA energy density is the (local) density, the GGA depends additionally on the gradient of the density, and the meta GGA has an extra dependency on the kinetic energy density.

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. This is not so important for a single point calculation as Band prints the bonding energies of a set of common functionals, but the *energy* functional is used for the nuclear gradients (geometry optimization). 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 (especially for Meta GGAs) 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 *(meta)GGA* to define the LDA and (meta)GGA parts of the functional. All subkeys are 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}
{DiracGGA  GGA}
{MetaGGA  {Apply}   GGA}
{Dispersion {s6scaling} {RSCALE=r0scaling} {Grimme3/dDsC} {BJDAMP}
{PAR1=par1} {PAR2=par2} {PAR3=par3} {PAR4=par4}
{Model} {LB94/TB-mBJ/GLLB-SC}}
End
```

The common use is to specify either an LDA, line or a (meta)GGA line. (Technically it is possible to have an LDA line *and* a GGA line, in which case the LDA part of the GGA functional (if applicable) is replaced by what is specified by the LDA line.)

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

`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]
`mPWx`: the modified PW91 exchange correction proposed in 1998 by Adamo-Barone [27]
`PBEx`: the exchange correction proposed in 1996 by Perdew-Burke-Ernzerhof [12]
`HTBSx`: the HTBS exchange functional [45]
`RPBEx`: the revised PBE exchange correction proposed in 1999 by Hammer-Hansen-Norskov [13]
`revPBEx`: the revised PBE exchange correction proposed in 1998 by Zhang-Wang [28]
`mPBEx`: the modified PBE exchange correction proposed in 2002 by Adamo-Barone [29]
`OPTX`: the OPTX exchange correction proposed in 2001 by Handy-Cohen [30]

For the correlation part the options are:
`Perdew`: the correlation term presented in 1986 by Perdew [6].
`PBEc`: the correlation term presented in 1996 by Perdew-Burke-Ernzerhof [12] .
`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:
`BP86`: this is equivalent to Becke + Perdew together.
`PW91`: this is equivalent to pw91x + pw91c together.
`mPW`: this is equivalent to mPWx + pw91c together.
`PBE`: this is equivalent to PBEx + PBEc together
`HTBS`: this is equivalent to HTBSx + PBEc together
`RPBE`: this is equivalent to RPBEx + PBEc together
`revPBE`: this is equivalent to revPBEx + PBEc together
`mPBE`: this is equivalent to mPBEx + PBEc together
`BLYP`: this is equivalent to Becke (exchange) + LYP (correlation).
`OLYP`: this is equivalent to OPTX (exchange) + LYP (correlation).
`OPBE`: this is equivalent to OPTX (exchange) + PBEc (correlation) [31].

`DiracGGA`

(Expert option.) Sometimes it is better to use the LDA potential for the Numerical atoms, when the exact solutions are too wild. We found this to be the case for the PBE functional (and its variations). In that case the matrix elements of the Kinetic energy operator converged very slowly with the Accuracy parameter.

`MetaGGA`

Key to select the evaluation of a meta GGA. A byproduct of this option is that the bonding energies of all known functionals are printed (using the same density). Meta GGA calculations can be quite a bit more time consuming, especially when active during the SCF.

Self consistency of the meta GGA is implemented as suggested by Neuman, Nobes, and Handy. [11]

The available functionals of this type are:
`M06L`: The Meta GGA as developed by the Minesota group [16]
`TPSS`: The 2003 Meta GGA [15]
`revTPSS`: The 2009 revised Meta GGA [26]

`DISPERSION Grimme3 BJDAMP`

If `DISPERSION Grimme3` is present a dispersion correction (DFT-D3-BJ) by Grimme [44] will be added to the total bonding energy, gradient and second derivatives, where applicable. This is the latest

dispersion correction in the DFTB-D family. Parametrizations are implemented for the functionals: B3LYP, TPSS, BP86, BLYP, PBE, PBEsol, and RPBE. It has four parameter. One can overide this using *PAR1=.. PAR2=..*, etc. In the table the relation is shown between the parameters and the real parameters in the Dispersion correction.

variable	variable on Bonn website
PAR1	s6
PAR2	a1
PAR3	s8
PAR4	a2

`DISPERSION Grimme3`

If `DISPERSION Grimme3` is present a dispersion correction (DFT-D3) by Grimme [42] will be added to the total bonding energy, gradient and second derivatives, where applicable. Parametrizations are available for: B3LYP, TPSS, BP86, BLYP, revPBE, PBE, PBEsol, and RPBE, and will be automatically set if one of these functionals is used. For all other functionals, PBE-D3 parameters are used as default. You can explicitly specify the three parameters

variable	variable on Muenster website
PAR1	s6
PAR2	sr,6
PAR3	s8

`Dispersion`

If the `DISPERSION` keyword is present a dispersion correction will be added to the total bonding energy, where applicable. By default the correction of Grimme [32]. The term is added to the bonding energies of all printed functionals, standard the LDA and a couple of GGAs. The global scaling factor with which the correction is added depends on the exchange-correlation functional used at SCF but it can be modified using the *s6scaling* parameter. The following scaling factors are used (with the XC functional in parantheses): 1.20 (BLYP), 1.05 (BP), 0.75 (PBE), 1.05 (B3LYP). In all other cases a factor 1.0 is used unless modified via the s6scaling parameter.
The van der Waals radii used in this implementation are hardcoded. However, it is possible to modify the global scaling parameter for them using the *RSCALE=r0scaling* argument. The default value is 1.1 as proposed by Grimme [32].
With the extra option `Grimme3` the latest dispersion correction, known as D3, will be used. [42]

`Model`

Some functionals give only a potential and have no energy expression. We call such functionals model potentials. In band there are two available. With `Model LB94` the asymptotically correct potential of van Leeuwen and Baerends is invoked. [10] To fix the band gap in bulk systems you can use `Model TB-mBJ`, see ref [46]. This potential depends on a c-factor for which there is a density dependent automatic

expression. However you can override the automatic value by specifying `XC%TB_mBJCFactor cfac`. In principle: the bigger the value the larger the gap. From a theoretical point of view the the GLLB-SC is the best, because it models the derivative discontinuity [51].

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).

Spin polarization

`Unrestricted`

If this key occurs (no additional data needed), a spin-*un*restricted calculation will be carried out.

```
UNRESTRICTED { Reference }
```

`Reference`

Default the formation energy is calculated with respect to the spherically symmetric spin-*restricted* atoms. If you want to do an *un*restricted calculation for the atoms, you may include this keyword as argument to key `Unrestricted`. This will only yield the true ground state density for atoms with a spherical (spin polarized) ground state like H, Li, N, Cr, Cu, ... In other words: this key does not provide a general solution for atomization energies.

Defaults and special cases
A calculation is default *restricted*.
General remarks
Be aware that cpu-time may be approximately doubled!

GGA+U

A special way to treat correlation is with so-called LDA+U, or GGA+U calculations. It is intended to solve the band-gap problem of traditional DFT, the problem being an underestimation of band gaps for transition-metal complexes. A Hubbard like term is added to the normal Hamiltonian, to model on-site interactions. In its very simplest form it depends on only one parameter, U, and this is the way it has been implemented in band. The energy expression is equation (11) in the work of Cococcioni [49]. See also the review article [48]

`HubbardU (block-type)`

Key to control the GGA+U calculation

```
HubbardU
          Enabled {true | false}
          PrintOccupations {true | false}
          UValue u1, u2, ...
          LValues l1, l2, ...
        End
```

`Enabled`: Whether or not to apply the Hubbard Hamiltonian

`PrintOccupations`: Whether or not to print the occupations during the SCF.

`UValue`: For each atom type specify the U value. A value of 0.0 is interpreted as no U

`LValue`: For each atom type specify the l value. A negative value is interpreted as no l-value

A typical example to apply LDA+U to the d-orbitals of NiO looks like

```
HubbardU
    printOccupations true
    Enabled true
    uvalue 0.3 0.0
    lvalue 2 -1
    End

    ATOMS Ni
    0.0 0.0 0.0
    END

    ATOMS O
    2.085 2.085 2.085
    END
```

Relativistic Effects

Relativistic effects are treated with the accurate and efficient ZORA approach [17-18], controlled by the `Relativistic` keyword. Relativistic effects are negligible for light atoms, but grow to dramatic changes for heavy elements. A rule of thumb is: Relativistic effects are quite small at row 4, but very large at row 6 (and later).

`Relativistic`

Includes a relativistic correction in the Hamiltonian.

```
Relativistic {level}
```

`Level {None | ZORA | ZORA Spin}`

`NONE`: no relativistic effects

`ZORA`: scalar relativistic ZORA. This option comes at very little cost, and is advised when you want to compare to experimental results.

`ZORA Spin`: Spin-orbit coupled ZORA. This is the best level of theory, but it is much more (4-8 times) expensive than a normal calculations. Spin-orbit effects are generally quite small, unless there are very heavy atoms in your system, especially with p valence electrons (like Pb).

Solvation

COSMO: Conductor like Screening Model

You can study chemistry in solution, as contrasted to the gas phase, with the implementation in BAND of the Conductor like Screening Model (COSMO) of solvation [36].

In the COSMO model all solvents are roughly the same, and approximated by an enveloping metal sheet. One explicit dependency on the solvent is that the solvation energy is scaled by

$f(\varepsilon) = (\varepsilon-1)/(\varepsilon+x)$

and this depends on the dielectric constant of the solvent, and an empirical factor *x*. The other is that shape of the surface is influenced by the RAD parameter, see below.

```
SOLVATION
 {Surf Esurf}
 {Solv {Name=solvent} {Eps=78.4}{Rad=1.4} {Del=1.4}
       {Emp=0.0} }
 {Div {Ndiv=3} {NFdiv=1} {Min=0.5} {OFAC=0.8}
      {leb1=23} {leb2=29}{rleb=1.5} }
 {RADII
 name1=value1
 name2=value2
  ...
 subend }
 {Charged {Method=meth} {Conv=1e-10} (Iter=1000} {NoCorr} }
 {Cvec {Exact | FitPot}
 {SCF {Var | Pert}
End
```

Presence of the SOLVATION key block triggers the solvent calculation and does not require additional data. With subkeys you can customize various aspects of the model, for instance to specify the type of solute. None of the subkeys is obligatory

See also the key `GeoOpt%StaticCosmoSurface` and `Frequencies%StaticCosmoSurface`.

Follows a description of the subkeys

`Surf {Wsurf|Asurf|`**`Esurf`**`|Klamt|Delley}`

Five different cavity types are available. The Wsurf, Asurf, and Esurf surfaces are constructed with the GEPOL93 algorithm [38].

`Wsurf`

Wsurf triggers the Van der Waals surface (VdW), which consists of the union of all atomic spheres.

`Asurf`

Asurf gives the Solvent-Accessible-Surface (SAS). This is similar to VdW but consists of the path traced by the center of a spherical solvent molecule rolling about the VdW surface or, equivalently, a VdW surface created by atomic spheres to which the solvent radius has been added. These two surface types contain cusps at the intersection of spheres.

`Esurf`

Esurf (the default) gives the Solvent-Excluding-Surface (SES), which consists of the path traced by the *surface* of a spherical solvent molecule rolling about the VdW surface. Primarily, this consists of the VdW surface but in the regions where the spheres would intersect, the concave part of the solvent sphere replaces the cusp. This SES surface is the default.

`Klamt`

The fourth surface option is Klamt as described in [39]. It excludes the cusp regions also.

`Delley`

The fifth surface is the so called Delley surface, see also Ref. [40].

`Solv`

Solvent details.

`Name, Eps, Rad`

Eps specifies the dielectric constant (the default relates to water). An infinite value for Eps is chosen if Eps is specified to be lower than 1.0. Rad specifies the radius of the (rigid sphere) solvent molecules, in angstrom. Instead of specifying Eps and Rad one can specify a solvent name or formula after 'name='. The following table lists names and formulas that are recognized with the corresponding values for Eps and Rad. The names and formulas are case-insensitive.

Name	Formula	Eps	Rad
AceticAcid	CH3COOH	6.19	2.83
Acetone	CH3COCH3	20.7	3.08
Acetonitrile	CH3CN	37.5	2.76
Ammonia	NH3	16.9	2.24
Aniline	C6H5NH2	6.8	3.31
Benzene	C6H6	2.3	3.28
BenzylAlcohol	C6H5CH2OH	13.1	3.45
Bromoform	CHBr3	4.3	3.26
Butanol	C4H9OH	17.5	3.31
isoButanol	(CH3)2CHCH2OH	17.9	3.33
tertButanol	(CH3)3COH	12.4	3.35
CarbonDisulfide	CS2	2.6	2.88
CarbonTetrachloride	CCl4	2.2	3.37
Chloroform	CHCl3	4.8	3.17
Cyclohexane	C6H12	2	3.5
Cyclohexanone	C6H10O	15	3.46
Dichlorobenzene	C6H4Cl2	9.8	3.54
DiethylEther	(CH3CH2)2O	4.34	3.46
Dioxane	C4H8O2	2.2	3.24
DMFA	(CH3)2NCHO	37	3.13
DMSO	(CH3)2SO	46.7	3.04
Ethanol	CH3CH2OH	24.55	2.85

EthylAcetate	CH3COOCH2CH3	6.02	3.39
Dichloroethane	ClCH2CH2Cl	10.66	3.15
EthyleneGlycol	HOCH2CH2OH	37.7	2.81
Formamide	HCONH2	109.5	2.51
FormicAcid	HCOOH	58.5	2.47
Glycerol	C3H8O3	42.5	3.07
HexamethylPhosphoramide	C6H18N3OP	43.3	4.1
Hexane	C6H14	1.88	3.74
Hydrazine	N2H4	51.7	2.33
Methanol	CH3OH	32.6	2.53
MethylEthylKetone	CH3CH2COCH3	18.5	3.3
Dichloromethane	CH2Cl2	8.9	2.94
Methylformamide	HCONHCH3	182.4	2.86
Methypyrrolidinone	C5H9NO	33	3.36
Nitrobenzene	C6H5NO2	34.8	3.44
Nitrogen	N2	1.45	2.36
Nitromethane	CH3NO2	35.87	2.77
PhosphorylChloride	POCl3	13.9	3.33
IsoPropanol	(CH3)2CHOH	19.9	3.12
Pyridine	C5H5N	12.4	3.18
Sulfolane	C4H8SO2	43.3	3.35
Tetrahydrofuran	C4H8O	7.58	3.18
Toluene	C6H5CH3	2.38	3.48
Triethylamine	(CH3CH2)3N	2.44	3.81
TrifluoroaceticAcid	CF3COOH	42.1	3.12
Water	H2O	78.39	1.93

`Del`

Del is the value of Klamt's delta_sol parameter, only relevant in case of Klamt surface.

`Emp`

Emp addresses the empirical scaling factor *x* for the energy scaling.

`DIV`

`Ndiv, NFdiv`

Ndiv controls how fine the spheres that in fact describe the surface are partitioned in small surface triangles, each containing one point charge to represent the polarization of the cavity surface. Default division level for triangles Ndiv=3. Default final dision level for triangles NFdiv=1 (NFdiv≤Ndiv).

`Min`

Min specifies the size, in angstrom, of the smallest sphere that may be constructed by the SES surface. For VdW and SAS surfaces it has no meaning. Default Min=0.5

`Ofac`

Ofac is a maximum allowed overlap of new created spheres, in the construction procedure. Default Ofac=0.8.

`leb1, leb2, rleb`

For the Delley type of construction one needs to set the variables `leb1` (default value 23), `leb2` (default value 29), and `rleb` (default value 1.5 Angstrom) to set the number of surface points. If the cavity radius of an atom is lower than `rleb` use `leb1`, otherwise use `leb2`. These values can be changed: using a higher value for `leb1` and `leb2` gives more surface points (maximal value `leb1`, `leb2` is 29). A value of 23 means 194 surface points in case of a single atom, and 29 means 302 surface points in case of a single atom Typically one could use `leb1` for the surface point of H, and `leb2` for the surface points of other elements.

`Radii`

In the Radii data block you give a list of atom types and values.

```
SOLVATION
  ...
 Radii
  H 0.7
  C 1.3
  ...
 Subend
  ...
End
```

The values are the radii of the atomic spheres, in the default unit of length (angstrom or bohr). If not specified the default values are those by Allinger. [41]

`Charge`

This addresses the determination of the (point) charges that model the cavity surface polarization. In COSMO calculations you compute the surface point charges *q* by solving the equation A*q*=-*f*, where *f* is the molecular potential at the location of the surface charges *q* and A is the self-interaction matrix of the charges. The number of charges can be substantial and the matrix A hence very large. A direct method, i.e. inversion of A, may be very cumbersome or even impossible due to memory limitations, in which case you have to resort to an iterative method.
Meth specifies the equation-solving algorithm. Meth=INVER (default) requests direct inversion. Meth=CONJ uses the preconditioned biconjugate gradient method. This is guaranteed to converge and does not require huge amounts of memory.
CONV and ITER are the convergence criterion and the maximum number of iterations for the iterative method.

In periodic calculations the unit cell needs to be neutral. Therefore, by default the COSMO charges are forced to sum to zero. When NOCORR is present, this constraint is not applied.

`CVec`

There are two ways to calculate the molecule- (COSMO)surface interaction.
EXACT (default): Consider the interactions between the electron density and the potential due to the COSMO points.
FITPOT: Use the COSMO point charges in the potential of the molecule (obtained by density fitting).

The EXACT option is exact on paper, but in reality numerical integration is used to get the energy. The standard grid is formally not suitable to handle the cusps at the COSMO points. Still this option works well because the singularity is softened by the smallness of the electronic density at the COSMO points. The EXACT option is much faster when doing a geometry optimization.

`SCF`

The default (VAR) is to apply the external potential of the COSMO surface during the SCF. To apply COSMO only post SCF you can use the PERT option.

Static electric field

For systems not having 3D periodic symmetry it is possible to specify an external electric field in the z-direction.

`EField`

Include a homogeneous static electric field in the z-direction.

```
EField
Ez ez
Unit {a.u.| Volt/bohr | Volt/Angstrom | Volt/Meter}
End
```

and here *ez* is the electric field in the unit specified by *unit*. The default unit is Volt/Angstrom.

The static field is explicitly handled in the determination of the orbital coefficients, energy, and gradients. If you apply it to any other property, such as the NMR shielding tensor, dielectric function, solvation energy, etc., the result is probably not entirely correct. In case of doubt contact the SCM staff.

Finite nucleus

Normally the nucleus is modelled as a point charge.

`NuclearModel`

Specify what model to use for the nucleus.

```
NuclearModel {PointCharge | Gaussian | Uniform}
```

and here one can choose between the standard point charge model and a Gaussian model. For the Gaussian model the nuclear radius is calculated according to the work of Visscher and Dyall [52].

The static field is explicitly handled in the determination of the orbital coefficients, energy, and gradients. If you apply it to any other property, such as the NMR shielding tensor, dielectric function, solvation energy, etc., the result is probably not entirely correct. In case of doubt contact the SCM staff.

2.5 Precision and Performance

The three main issues that influence the accuracy of a calculation are: integration in real space, integration in k-space, and the quality of the basis set.

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). In the unlikely event that this single parameter does not suffice, you can fine-tune the integration scheme, see the description of the `Integration`. key-block of the ADF documentation. The other major key influencing the integration in reciprocal space is the `KSpace` parameter.

General accuracy parameter

`Accuracy`

General precision parameter. Default value = 3.5, which is a bit low. For your final calculation in general a value of 4.5 is advised. Higher values (larger than 4.5) should only be necessary in extreme cases. The tell-tale sign of a too low value is that a binding curve becomes bumpy. 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.

Fine tuning integration in real space

Many of the integrals are obtained by numerical integration. Two grids are available: the old Voronoi scheme (the default) and the new Becke grid. Once can switch between the two using:

```
IntegrationMethod {Voronoi | Becke}
```

Voronoi grid

`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.

Becke grid for numerical integration

The Becke grid is a refined version of the fuzzy cells integration scheme developed by Becke, see Ref. [53]. The Becke grid is better suited for geometry optimization and TS search compared to the Voronoi scheme. The Becke integration scheme leads to a smoother behavior of the relative integration error as a function of the nuclear coordinates, which consequently results in more reliable optimizations and higher achievable convergence. The Becke grid is not very well suited to calculate Voronoi deformation density (VDD) charges. For accurate calculation of VDD charges the Voronoi integration scheme is recommended.

The default quality of the Becke grid is normal. It can be changed with the block key BECKEGRID:

```
IntegrationMethod Becke

BeckeGrid
 Quality {basic|normal|good|verygood|excellent}
End
```

A Becke grid of normal quality is roughly equivalent (in both absolute accuracy and computation time) to INTEGRATION 4 (Voronoi scheme), and a Becke grid of good quality is roughly equivalent to INTEGRATION 6 (Voronoi scheme).

Numerical integration in reciprocal space

`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 `kspace`-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 `Kspace`. For other settings, see the explanation of `Kspace` in the second list. Default depends on `Accuracy`, the integration parameter in real space.

Using a regular grid

```
KSpace
  Grid n1 n2 n3
End
```

With this subkey you will get a simple regular grid in k-space. There are two benefits: you can make the sampling direction dependent (if one of the reciprocal lattice vectors is much shorter you can use fewer points in that direction). The second is that the number of k-points will always be the same, which may be useful during geometry optimizations with lattice optimization.

The normal method samples the irreducible wedge of the Brillouin zone, whereas the regular grid samples the whole BZ. As a rule of thumb you need to choose roughly twice the value. For example kspace 2 corresponds to grid 4 4 4, kspace 3 to grid 5 5 5, etc.. Sticking to this rule the number of unique k-points will be roughly similar.

Basis and fit set

With the key `BasisDefaults` you can control the quality of the basis set, and the level of frozen core approximation. Given your preferences the basis and fit sets are retrieved from the database. The output starts with a copy of your input, but with the basis and fit sets inserted from the database: there you can exactly see what values are used. You can always override the defaults by specifying the `AtomType` keyword. There is no need for special basis sets for relativistic calculations.

Using the database: BasisDefaults

`BasisDefaults (block-type)`

This key allows you to generate automatically the required `AtomType` blocks (i.e. basis and fit sets) from the database. The database is located in "$ADFHOME/atomicdata/Band". *You can always see what has been generated at the beginning of the output file, where your input is echoed with the AtomType blocks expanded.* It shows as a comment from which file each `AtomType` has been copied.

```
! --- basis from file /path/to/TZ2P/C.1s---

AtomType C
....
End
! --- basis from file /path/to/DZ/H---

AtomType H
```

```
....
End
```

The block has the following sub keys.

`BasisType`

The following basis sets are available.

- DZ: Double zeta. Adapted from the ADF basis set of the same name. The smallest basis set. The results are most likely not so accurate
- V: the original band basis set. Similar to the TZ2P basis set.
- TZ2P: Adapted from the ADF basis set of the same name. This is a good basis set.
- QZ4P: Quadruple zeta plus quadrupole polarization. The biggest basis set. For benchmark purposes only.

If you have a large unit cell, you could start with the DZ basis, to see if it will be feasible at all. For reliable results, use the V or the TZ2P basis. You can override the defaults per type by specifying `Atoms%Core` and `Atoms%BasisType`, or `Atoms%Path` (sub)keys.

`FitType`

Normally the fit set is determined by the BasisType. With this key can choose the fit independently from the BasisType key.

`Core`

This influences the size of the frozen core used. Possible values are "None", "Small", "Medium", "Large". You should be aware that a change of this parameter does not necessarily mean that a different frozen core is used. You can again check in the output what file was really used. You can see it from the extension in the comment line preceding the `AtomType` key

```
! --- basis from file /path/to/TZ2P/C.1s---
```

or from the `Dirac` block in the `AtomType` key.

Manually specifying AtomTypes

`AtomType (block-type)`

Description of the atom type. (For convenience you might prefer to generate them automatically with the `BasisDefaults` key.) 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 (i.e. the numerical atomic orbitals, see key `Dirac`).

`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
```

`TestFunctions (block-type)`

An optional subkey of the `AtomType` key block is `TestFunctions` that has 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. For the time being the *l* value is ignored. A possible application is to include a very tight function, to increase the accuracy near a nucleus.

Confinement of basis functions

It is possible to alter the radial part of the basis functions in order to make them more compact.

Hard confinement

`CONFINE (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.

Soft confinement

`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*'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.

NB: does not work with the `BasisDefaults` key (because it is a sub key of `AtomType`).

Technical Settings*

There are of course many other settings influencing the precision and performance. Usually the user does not need to care about them.

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 look at the subkeys `Mixing`, and `Degenerate`, and the subkey block `DIIS` .

SCF key

`SCF`

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

`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, ...).

`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.

Convergence key

`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`

`ElectronicTemperature`

(Slabs only) 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).

`SpinFlip`

List here the atoms for which you want the initial spin polarization to be flipped. This way you can distinguish between ferromagnetic and anti ferromagnetic states. Currently, it is now allowed to give symmetry equivalent atoms a different spin orientation. To achieve that you have to break the symmetry.

`StartWithMaxSpin` ***{true**|**false}***

To break the initial perfect symmetry of up and down densities there are two strategies. One is to occupy the numerical orbitals in a maximum spin configuration. The alternative is to add a constant to the potential. The default is *true*. See also `SCF%vsplit`.

`InitialDensity` ***{rho**|**psi}***

The SCF is started with a guess of the density. There are the following choices
RHO: the sum of atomic density.
PSI: construct an initial eigensystem by occupying the atomic orbitals. The guessed eigensystem is orthonormalized, and from this the density is calculated.

DIIS key

`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

clarge: 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 dimix*value + (1-dimix)*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 key*

```
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, which constructs the radial tables for the NAOs.

Linear Scaling

```
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) {fit crfit}
```

One real argument for keys basis and core, which should be a small value; default *crbas*=1e-6. The core criterion defaults to the bas criterion (if set). 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. Also for the fit one can specify a cutoff, but this is turned of by default: *crfit=1e-15*. Here we advise to use a more strict criterion than for crbas (because the fit is usually much more dependent than the basis). This option has some refinements. For example

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

As you see there are two new elements ('confine' and 'Rosa'). 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 (Rosa) 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 "Rosa" key works as follows. Determine the radius R where according to the normal criterion the function is negligible. Next substitute R -> R + 3*exp(-R/3). This means that for tight functions the tails radius is replaced by 3, and for diffuse functions this modifier has no effect.

Tentatively, based on the work by Rosa Bulo, we suggest to use

```
Tails bas=1e-3 Rosa
```

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.

Dependency

`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-8. 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-coreval(...)`. Default=1e-5.

Screening

The program BAND performs many lattice summations which are in practice truncated. The two prime examples are the construction of the Bloch basis (and also the fit) and the construction of the Coulomb potential of the fit functions. The precision of the lattice summations is controlled by the `SCREENING` key

`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 `rcelx`. 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`, `rcelx`. 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.

Fit Method

With this keyword the fit method can be influenced. Normally band tries to guess the best choice.

```
FitMethod
  Method {Auto | Inverse | Divide | CONJUGATEGRADIENT}
  Divide {lvalue | atom | pair | cluster}
  UseSharedArrays {true | false}
  ConstrainWithS {true | false}
  RScale rscale
End
```

`Method`: With *Auto* band will do what it thinks is best. With *Invserse* global fitting is enforced. The choice *Divide* invokes the Divide and Fit method. Finally, setting it to *CONJUGATEGRADIENT* the global fit will be used without inverting the overlap matrix.

`Divide`: Applies to Divide and fit method. *LVALUE*: split fit set per l-value, going from high to low l. *ATOM:* divide per atom. *PAIR:* divide per pair. *CLUSTER:* divide in overlapping cluster. This is the most accurate choice. The size of the cluster is controlled by RSCALE

`UseSharedArrays`: Whether or not to use shared arrays. Works only with Divide option.

`ConstrainWithS`: Works only with Divide option. The charge constraint is always done with the last division. With this option the set of s functions is added as the last division.

`RScale`: Works only with Divide option. Controls the size of the cluster. Default value is 1.3.

Orthogonal Fit

`ORTHOGONALFIT`

Transform the fit explicitly to an orthogonal basis. This is expensive, but is numerically more stable. This can be of use when there is a nearly dependent fit set. A better solution is to reduce the fit set.

Direct (on the fly) calculation of basis and fit

Band normally writes the basis functions (and derivatives if needed), and the fit functions (and potential plus derivatives if need) to disk. Usually it pays of to do this because calculating them is slower than reading them. However, whether or not that is the case depends strongly on the type of calculation and on the type of hardware. Here follow some considerations.

Systems: Basically what we need to look at is the number of cells that needs to be summed over. For chains (1D periodicity) there are few, and for bulk (3D) there are many. With a bigger unit cell, the number of cells also decreases. Fewer cells are in favor of the direct option.

Hardware: Many cores and slow disks are in favor of the direct option.

You can invoke the direct option like this

```
Programmer
DirectBas true
DirectFit true
End
```

Radial grid

With this keyword the radial grid can be controlled.

```
RadialDefaults
  NR nr
  RMin rmin
  RMax rmax
End
```

The number of radial points is *nr* (default=3000). With very high values like 30.000 the Dirac subprogram may not converge. The logarithmic grid runs from *rmin* (default 1e-6) to *rmax* (default 100)

Elliptic integrals

The integration of the electrostatic interaction between spherical atoms is done very precisely in an elliptic grid, which depends on `NUELSTAT` and `NVELSTAT`

`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

Fermi energy search

`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.

`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

IO settings*

IO block size

`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.

File limit

`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!

Block size

Efficiency and memory usage depend on how large the vector size of the program is

`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.

`KGRPX`

is an absolute upper bound on the number of k-points processed together. Specifying this key *and* `CPVector` you can override bands defaults.

2.6 Restarts

The main results of a band calculation are stored in the RUNKF file. If you save this file you can use it to restart your calculation. The input for the restart calculation is essentially the same, except for some extra keys, like `Restart`, `Grid`, and `DensityPlot`

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`. Likewise, plots of the density (and many other symmetric properties) can can be obtained with the key `DensityPlot`. Density and orbital plot restarts require the specification of the `Grid` key. With the subkey `SCF` you can start the SCF procedure with the last solution from the restart file. This can be useful if the SCF did not converge. Also if you have changed some settings that should have little effect, like the integration accuracy, the restart SCF option can help you to save a few cycles. Similarly, a geometry optimization can be restarted with the subkey `GeometryOptimization` You can use the geometry of a previous calculation with the `GeometryOptimization` subkey.

Restart key

`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, GeometryOptimization, Geometry, DOS, OrbitalPlot, or DensityPlot). See also keys `OrbitalPlot` and `DensityPlot`. Usually the input for a restart is the same as for the original calculation, where you add some extra options.

Plots of the density, potential, and many more properties

`DensityPlot (block-type)`

Goes together with the `Restart%DensityPlot` and `Grid` options.
Example input

```
...
Restart my.runkf &
DensityPlot
End

Grid
Type Coarse
End

DensityPlot
FITDENSITY_total_scf
XCPOTENTIAL_scf
End
...
```

After such a run you get a TAPE41 file that you should rename to my.t41, and view with adfview.

The most common properties to plot are:
`FITDENSITY_total_scf`: The fitted density.
`COULOMBPOTENTIAL_scf`: The Coulomb potential.
`XCPOTENTIAL_scf`: the XC potential (using the fitted density)
`XCPOTENTIAL_scf_exact`: XC potential of the exact density
`DENSITY_total_scf`: The density
`DENSITY_agrad_total_scf`: The norm of the gradient of the density
`TAU_total_scf`: The symmetric kinetic energy density
`LDOS_scf`: The local density of states (at the fermi level). The bias can be controlled with the LDOS block
`ELF_total_scf`: The electron localization function

Some more specialized options are:
`FITDENSITY_deformation_scf`: the fitted deformation density
`ATOMIC_density`: The density of the startup atoms
`ATOMIC_coulombPot`: The Coulomb potential of the start density

In the band example directory there is the Frags_COCu example that shows how this can be used in combination with the fragment option.

Orbital plots

`OrbitalPlot (block-type)`

Goes together with the `Restart%OrbitalPlot` and `Grid` options. In the band example directory there is the Cu_slab example that shows how this can be used. Example input

```
...
Restart my.runkf &
OrbitalPlot
End

Grid
Type Coarse
End

OrbitalPlot
```

```
1 Band 5 8 ! k-point 1 bands 5 to 8
5 Band 6   ! k-point 5 band 6
6 -0.2 +0.3 ! k-point 6 bands between -0.2 and +0.3 Hartree from the
fermi level
End
...
```

After such a run you get a TAPE41 file that you should rename to my.t41, and view with adfview.

Grid

`Grid (block-type)`

Used for restart options `Restart%OrbitalPlot`, and `Restart%DensityPlot`. There are two ways to define your grid. The most easy way is to use the `Type` key which can have the values `Coarse, Medium, and Fine`, for example

```
Grid
Type Coarse
End
```

This generates automatically a grid around the atoms in the unit cell. The alternative is to specify everything by hand. The following input would create a cube from (-1,-1,-1) to (1,1,1)

```
Grid
-1 -1 -1 ! Starting point
1 0 0 0.1 ! vec1 and dvec1
0 1 0 0.1 ! vec2 and dvec2
0 0 1 0.1  ! vec3 and dvec3
20 20 20 ! nr. of steps along three directions
End
```

LDOS (STM)

`LDOS (block-type)`

Local Density-Of-States information. This can be used to generate STM images in the Tersoff-Hamann approximation [35].

```
LDOS
  { Shift eshift }
  { DeltaNeg deneg }
  { DeltaPos depos }
End
```

`Shift`: The energy bias with respect to the fermi level in a.u.. Default = 0.0

`DeltaNeg` Default = 1e-4 a.u.

`DeltaPos` Default = 1e-4

The local density of states is integrated over the interval *eshift-deneg* through *eshift+depos*.
Example of an LDOS restart:

```
...
Restart my.runkf &
 DensityPlot
End

Grid
Type Coarse
End

DensityPlot
LDOS_scf
End

LDOS
 Shift      0.1
 DeltaNeg .001
 DeltaPos 0
End
```

According to this example, we restart from the result file of a previous calculation. The calculation will generate a file TAPE41 which can be viewed with adfview. (Rename the file to my.t41)

See also `Restart`, and `DensityPlot`.

Complete example scripts for visualisation

Below follows a complete example how to use band for visualising. We start with the normal calculation for e.g. a Li slab

```
#!/bin/sh
# Contents of the file LiSlab.job

"$ADFBIN/band" << eor
TITLE Li Slab

UNITS
    length Angstrom
    angle Degree
END

ATOMS Li
0.0 0.0 0.0
-1.745 -1.745 -1.745
END

Lattice
 3.49 0.000000 0
 0.000000 3.49 0
End

BasisDefaults
BasisType DZ
Core Large
End
```

```
XC
LDA SCF VWN
END

end input
eor

# store the result file
mv RUNKF LiSlab.runkf
```

Note that we store the result file in "LiSlab.runkf". In the next run we use this file to generate the plot file. The job is essentially a copy of the first job with some extra lines. In this case we instruct the program to generate plot information for the density and the LDOS

```
#!/bin/sh
# Contents of the file LiSlab_restart.job


"$ADFBIN/band" << eor
TITLE Li Slab

UNITS
    length Angstrom
    angle Degree
END


Restart LiSlab.runkf &
DensityPlot
End

Grid
Type Coarse
End

DensityPlot
FITDENSITY_total_scf
LDOS_scf
End

LDOS
shift 0.0
End

ATOMS Li
0.0 0.0 0.0
-1.745 -1.745 -1.745
END

Lattice
 3.49 0.000000 0
 0.000000 3.49 0
End

BasisDefaults
BasisType DZ
Core Large
```

```
End

XC
LDA SCF VWN
END

end input
eor

# Store the result file, and more importantly, the plot file (.t41)
mv RUNKF LiSlab_restart.runkf
mv TAPE41 LiSlab.t41
```

And then you can visualize the result with adfview.

```
$ADFBIN/adfview LiSlab.runkf
```

It is most convenient to start adfview from the .runkf file rather than from the .t41.

2.7 Structure and Reactivity

To study structure and reactivity common tasks are geometry optimization, transition state search and a frequency run. A frequency run can be done to obtain a good initial Hessian for a geometry optimization, or to check whether a proper minimum or transition state has been found.

The basis for locating stationary points on the PES are the nuclear gradients. (The user may assume that the program does that automatically when needed.)

Nuclear energy gradients*

If you want the program to calculate the gradient of the energy with respect to nuclear displacements you should add the `Gradients` keyword. If the `SelectedAtoms` key is present, only the gradient of the selected atoms will be evaluated, and the rest set to zero.

References

Details about the theory of the analytical gradients within the BAND program can be found in the work of Kadantsev and coworkers [19]

`Gradients (block-type)`

Key to control the automatic geometry optimization.

```
Gradients
  oldio {0 | 1}
  LatticeStepSize step
  UseRelativeLatticeStep  (true | false)
End
```

`oldio`: There are two slightly different implementations. The *oldio=1* implementation has a better scaling behavior for large systems, although it can be a bit slower for small ones. When used with MetaGGAs *oldio=1* is required.(default = 1)

`latticeStepSize`: The lattice gradients are obtained by numerical differentiation. With this keyword the step size can be changed. (default = 0.012 bohr). When `UseRelativeLatticeStep` is set to true the step will be relative to the size of the displacement.

The user rarely needs to specify this key as it is automatically added for geometry optimizations and frequency runs.

This options honors the `SelectedAtoms` key, in which case only the forces will be calculated for the selected atoms.

Lattice gradients*

Band currently has a numerical implementation of the forces that act on the lattice vectors. The only way to invoke this is to do a geometry optimization, including lattice vectors, see the `GeoOpt%OptimizeLattice` key. The step size is controlled with `Gradients%latticeStepSize`, and `Gradients%UseRelativeLatticeStep`.

Geometry optimization

The geometry can be optimized by adding the `GeoOpt` key block. Within the block you can specify the maximum number of cycles and the convergence criterion. The optimizer works with Cartesian coordinates and a Quasi Newton stepper. The initial Hessian is by default the unit matrix, but can also be loaded from a previous geometry optimization or frequency run. Various constraints are possible such as bond distances and angles. The GDIIS convergence accelerator is available but is disabled by default. A geometry optimization can be restarted from a previous result (.runkf) file. The geometry can be optimized non-relativisticlly, with scalar relativistic ZORA, or with spin-orbit coupling.

GeoOpt key

`GeoOpt (block-type)`

Key to control the automatic geometry optimization.

```
GeoOpt
  OptimizeLattice {true | false}
  Iterations n
  Converge {Grad=tolgrad} {E=tolE} {Step=tolStep}
  TrustRadius radius
  UseVariableTrustRadius var
  InitialHessian filename | UnitMatrix| Swart
  RestartSCF {true | false}
  RestartFit {true | false}
  GDIIS {true | false}
  StaticCosmoSurface {true | false}
  TSMode mode
End
```

`OptimizeLattice`: Whether or not to optimize the lattice vectors. (default = false). The lattice gradients are obtained by numerical differentiation, see `GRADIENTS%LatticeStepSize`.

`Iterations`: the maximum number of cycles: default 50

`Converge {Grad=`*tolgrad*`} {E=`*tolE*`} {Step=`*tolStep*`}`: various criteria to determine the convergence can be specified on line. Here *tolGrad* is the maximum gradient allowed (default=1.0e-2 Hartree per Angstrom), *tolE* is the maximum energy change allowed (default 1.0e-3 Hartree), *tolStep* is the maximum step size (default 3e-2 bohr)

`TrustRadius`: The step sizes taken by the optimizer will be limited to this value (default = 0.2 bohr). If the proposed step is larger it will be scaled back.

`UseVariableTrustRadius`: Automatic adjustment of the trust radius based on the observed energy changes. Default: false.

`InitialHessian`: The initial hessian is read from a "RUNKF" file *filename* of a previous frequency calculation. By default the unit Hessian will be used. Another option is to use the empirical force field derived Swart Hessian.

`RestartSCF`: During the optimization try to restart the SCF from the previous geometry step. (default = true)

`RestartFit`: During the optimization try to restart the SCF from fit coefficients of the previous geometry step. (default = false). Can not be used simultaneously with RestartSCF.

`GDIIS`: Use the GDIIS convergence accelerator (default = false)

Note that a (failed) geometry optimization can be continued with the `Restart` key by specifying as option `GeometryOptimization`.

`StaticCosmoSurface`: Keep the Cosmo surface (if any) constant. (default = false)

`TSMode`: Hessian normal mode index to follow during transition state search. The specified mode is selected on the first iteration and is followed regardless of its index on subsequent iterations. Modes with zero frequencies are not counted.

Numerical frequencies (Hessian)

By specifying the runtype `Frequencies` you can calculate the Hessian which in turn yields the harmonic frequencies and vibrational modes. The Hessian is an important product that can be used in a geometry optimization, and in particular to start a transition state search. The second derivative of the energy (i.e. the Hessian) is obtained by numerically differentiating the analytical gradients. When your unit cell has N atoms in the unit cell and no symmetry (other than translational symmetry) the number of displacements is 2x3xN. For large systems this can be very time consuming, and you could consider to calculate only a partial Hessian. If you have a very symmetric unit cell, but are interested in asymmetric modes, set the subkeys `useA1Displacements` and `doA1Projection` to false. NB: the lattice vectors are assumed to be fixed.

A frequency run is invoked with

```
RunType
Frequencies
End
```

The rest is controlled by the `Frequencies` key.

`Frequencies (block-type)`

Key to control the numerical frequency run

```
Frequencies
  Step size
  useA1Displacements {true | false}
  doA1Projection {true | false}
  StaticCosmoSurface {true | false}
End
```

`Step`: The step size (angstrom) for the numerical differentiation

`useA1Displacements`: Determine only the total symmetric modes (default = true). When disabled, NOSYM calculations will be performed.

`doA1Projection`: Symmetrize the Hessian (default = true). For most users the only sensible setting is to make it equal to the value of `useA1Displacements`.

`StaticCosmoSurface`: Keep Cosmo surface (if any) constant. Default = true. Although this is an approximation it is numerically much more stable. If you set it to *false* there is the risk that the number of Cosmo points is not constant for all finite displacements.

This options honors the `SelectedAtoms` key, in which case only the hessian will be calculated for the selected atoms.

Transition state search

A transition state search is technically a geometry optimization. It is a search in the 3xN space for a point with vanishing gradients. The only difference is that it will try to go uphill in the direction of the lowest vibrational mode. The normal procedure is to guess a point that is as close as possible to the transition state. In that point you do a frequency run, which should hopefully produce a lowest vibrational mode in the direction of the transition state. You then start the Transition state search from that point, using the hessian of the frequency run. At the end you can do again a frequency run to check that there is indeed a single negative vibrational mode.

The transition state is invoked like this

```
RunType
TS
End
```

The rest is controlled by the `GeoOpt` key as in a normal geometry optimization.

Partial Hessian and (pre)optimizations

If you consider the interaction of a molecule with a surface it may be initially convenient to freeze some of the lower layers in the slab, making calculations much cheaper. This can be done for the evaluation of the hessian (see `SelectedAtoms`) and during a geometry optimization or transition state search (see `Constraints`). When using a partial hessian to start a geometry optimization note this. The atoms that you select during the frequency run, are the ones that should not be frozen in the geometry optimization. In a way the selection needs to be inverted.

Constrained optimization

During a geometry optimization certain constraints can be enforced with the `Constraints` key. The constraints refer to the coordinates of the unit cell as specified on input.

`Constraints (block-type)`

Key to control constraints during a geometry optimization.

```
Constraints
  Atom a1 {x y z}
  Dist a1 a2 distance
  Angle a1 a2 a3 angle
  Dihed a1 a2 a3 a4 angle
End
```

`Atom`: Freeze the position of atom *a1* to the current coordinate, or if specified to (*x,y,z*). This constraint will make the calculation cheaper, because the gradients of frozen atoms need not be calculated.

`Dist`: Constrain the distance between atoms *a1* and *a2* to *distance*

`Angle`: Constrain the angle between atoms *a1*, *a2*, and *a3* to *angle*

`Dihed`: Constrain the dihedral angle between atoms *a1*, *a2*, *a3*, and *a4* to *angle*

Selected atoms

When doing a frequency run you can restrict it to a limited number of selected nuclei. The selection may not break the symmetry.

```
SelectedAtoms a1 a2 ... an
```

The list of atoms with indices *a1 a2 ... an* are considered selected. Currently this option only has effect in a numerical frequency run and with the `Gradients` keyword.

Phonons and thermodynamics

Atoms vibrate about their equilibrium positions, giving rise to lattice vibrations called phonons. Band can calculate phonon dispersion curves within standard harmonic theory, implemented with a finite difference method in the spirit of the program PHON, described in Ref. [47]. Within the harmonic approximation we can calculate the partition function and from that thermodynamic properties such as the specific heat and the free energy. Our implementation can also handle 1D an 2D periodic systems.

Essentially you need to do the following steps.

- Optimize the structure, including the lattice vectors. The resulting geometry is the starting geometry for the phonon run
- Specify a super cell transformation. In principle this should be as large as possible. In practice one may want to start with a 2x2x2 cell. Set the runtype to Phonon
- Examine with the GUI the dispersion curves, and check for negative frequencies. If these exist then either the geometry was not close enough to the minimum, or the precision of the phonon run was not enough.

PhononConfig

`PhononConfig (block-type)`

Key to control the phonon run.

```
PhononConfig
  SuperCell Sub block type
  StepSize step
  MinTempKelvin tmin
  MaxTempKelvin tmax
  NumTemperatures ntemp
End
```

`SuperCell`: This subblock key should hold the supercell transformation, which expresses the lattice vectors in terms of the vectors of the primitive cell.

`StepSize`: The step size taken to obtain all force constants.

`MinTempKelvin`: Minimum temperature for thermodynamic plots.

`MaxTempKelvin`: Maximum temperature for thermodynamic plots.

`NumTemperatures`: Number of temperatures for thermodynamic plots.

Here is an example input

```
phononConfig
  stepSize 0.0913
  superCell
    2 0 0
    0 2 0
    0 0 2
  subend
end
```

In the regular output the thermodynamic functions are printed

```
========================================
          Thermo Dynamic Properties
========================================
Zero-point Energy (hartree)         0.00448537
Zero-point Energy (eV)              0.12205313

=============================================================================================================
       Temperature(K) Internal energy(Hartree)          Entropy(kB)    Free Energy(Hartree)       Specific
Heat(kB)

        0.000000E+00         0.448537E-02         0.000000E+00          0.448537E-02
0.000000E+00
        0.100100E+01         0.448540E-02         0.116921E-01          0.448536E-02
0.329896E-01
        0.200200E+01         0.448566E-02         0.643295E-01          0.448525E-02
0.138532E+00
...
```

2.8 Spectroscopic properties

Time-dependent DFT

In this section, the time-dependent density functional theory implementation in BAND is described. How to do a response calculation in BAND (which input keys to use), can be found in the list of keywords, see the key Response. 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 $\varepsilon_e(\omega)$ These are mutually related,

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

In general $\chi_e(\omega)$ and $\varepsilon_e(\omega)$ are tensors, which, however, simplify to scalars in isotropic systems. The above formula is valid in the case of insulators and semiconductors, where the bands are either fully occupied or fully unoccupied. It is defined as the interband part of the dielectric function and it is due to transitions from occupied bands to unoccupied bands. Also in the metallic case similar expressions can be found, the main difference is that there now is also an intraband contribution. Some examples are available in the $ADFHOME/examples/band directory and are discussed in the Examples document.

References

The three related Ph.D. theses, due to F. Kootstra (on TD-DFT for insulators), P. Romaniello (on TD-CDFT for metals), and A. Berger (on the Vignale-Kohn functional in extended systems) contain much background information, and can be downloaded from the SCM website.

The most relevant publications on this topic due to the former "Groningen" group of P.L. de Boeij are [22-25]

Response key

`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
   ebndtl 0.001
   ifxc    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 $\varepsilon_e(\omega)$ 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 criterion for the change in the fitcoefficients for the fitfunctions, when fitting the density (default=1d-3).

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

`ebndtl` 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)),
Only the default option for ifxc is implemented at the moment.

`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

`static` An alternative method that allows an analytic evaluation of the static response (Normally the static component is approximated by a finite small value). This option should only be used only for (non relativistic calculations on) insulators, and it has no effect on metals. Experience shows that KSPACE convergence can be slower, and this is why it is not the default.

`newvk` Use the slightly modified version of the VK kernel (Phys. Rev. B 74, 245117). When using this option one uses effectively the >`static` option, even for metals, so one should check carefully the convergence with the KSPACE parameter. An example response block for a vignale kohn calculation looks like

```
Response
  nfreq   2
  strtfr  0.10d0
  endfr   0.25d0
  newvk
  iyxc    1
  cnt
 END
```

`qv/cnt` Use the QV or CNT parametrization for the longitudinal and transverse kernels of the xc-kernel of the homogeneous electron gas. Use this in conjunction with the newvk option. (Refs: PRB 65, 235121 (QV); J. Phys. Condens. Matter 9, L475 (CNT))

Limitations

The method has not been implemented for slabs, so it can only be used for chains and 3d bulk systems.

Time-dependent DFT for metals

For metals the interband part of the dielectric function is due to transitions from (partially)-occupied bands to (partially)-unoccupied bands. Now there is also a term, which is called the intraband part of the dielectric function, which is due to transitions within the same partially-occupied band. The macroscopic dielectric function $\varepsilon_e(\omega)$ is now calculated as

$\varepsilon_e(\omega)=1+4\pi\chi_e(\omega) - 4\pi i\sigma_e(\omega)/\omega$.

Convergence and reproducibility: For TD-DFT calculations on metals a dense sampling of reciprocal space is required, i.e. the results converge slowly with the KSPACE parameter. The dielectric function might even not reproduce well across different machines. (Usually results that are not yet converged with the KSPACE parameter, like the energy, are reproducible across different platforms). Nevertheless, when the KSPACE parameter is chosen sufficiently high, the same result will be obtained on all machines. For instance for Cu the machine dependence was less than 0.01 for the dielectric function with KSPACE=11. In short: check for the convergence of the dielectric function with respect to the KSPACE parameter.

Frequency dependent kernel

It is known that the exact Vignale Kohn kernel greatly improves the static polarizabilities of infinite polymers and nanotubes (see JCP 123 174910), but gives bad results for the optical spectra of semiconductors and metals. For the low frequency part one needs a frequency dependent kernel, because Drude-like tails are completely absent in the ALDA. With a modified Vignale Kohn kernel, neglecting μ_{xc} so that it reduces to the ALDA form in the static limit (see PRB 74 245117) much better results can be obtained. Band currently only supports the modified VK kernel in either the QV or CNT parametrization, and it should only be used for metals.

EELS

Once the macroscopic dielectric function is known it is possible to calculate the electron energy loss function (EELS). In transmission electron energy loss spectroscopy one studies the inelastic scattering of a beam of high energy electrons by a target. The scattering rates obtained in these experiments is related to the dynamical structure factor $S(\mathbf{q},\omega)$ [1]. In the special case with wavevector $\mathbf{q}=0$, $S(\mathbf{q},\omega)$ is related to the longitudinal macroscopic dielectric function. This is the long-wave limit of EELS.
For isotropic system the dielectric function is simply a scalar (1/3 Tr $\varepsilon_e(\omega)$). In this case the long-wave limit of the electron energy loss function assumes the trivial form

$\mathrm{Lim}(q\rightarrow 0)\ 2\pi\ S(\mathbf{q},\omega)/(q^2V) = \varepsilon_2/(\varepsilon_1^2+\varepsilon_2^2)$

with ε_1 and ε_2, respectively, the real and imaginary part of the dielectric function.

[1] S. E. Schnatterly, in Solid State Physics Vol.34, edited by H. Ehrenreich, F. Seitz, and D. Turnbull (Academic Press, Inc., New York, 1979).
[2] P. Romaniello, and P. L. de Boeij, Phys. Rev. B (accepted).

ESR

Band is able to calculate electron paramagnetic resonance parameters of paramagnetic defects in solids: hyperfine A-tensor and the Zeeman g-tensor.

The implementation of EPR parameters in BAND are described in the publications by Kadantsev and coworkers [20] and [21]

Hyperfine A-tensor

The A-tensor is implemented within the nonrelativistic and scalar relativistic spin-polarized Kohn-Sham scheme. The A-tensor calculation is invoked by block

```
ATENSOR
END
```

Also, note that "Unrestricted" keyword should be present.

Two methods are used for A-tensor calculation.

Method 1 involves gradient of spin-polarization density and integration by parts. The isotropic component of A-tensor is obtained through integration, in a "nonlocal fashion".

In Method 2, the A-tensor is computed from spin-polarization density. Method 2 does not relies on the integration by parts. The isotropic component is obtained in a "local fashion" from the value of spin-polarization density on the grid points near the nuclei.

The user should be aware that numerical integration in A- and g-tensor routines is carried out over Wigner-Seitz (WS) cell, and, therefore, to obtain a meaningful result, the defect in question should lie at or, very close to, WS cell origin. This might require, on the user's part, some modification of the input geometry.

It also might happen that the size of the WS cell is not large enough for the adequate description of the paramagnetic defect in question. In this case, Method 1, which relies on the integration by parts and assumes that the spin-polarization density is localized inside the WS cell will fail. For the same reason, We recommend that the user removes diffuse basis set functions that describe the defect subsystem.

Finally, we note that the final result for A-tensor as presented by BAND is not scaled by the nuclear spin (as it is done in ADF) and the user is responsible for making neccessary adjustments.

g-tensor

The calculation of Zeeman g-tensor is invoked with block

```
ESR
END
```

The Zeeman g-tensor is implemented using two-component approach of Van Lenthe and co-workers in which the g-tensor is computed from a pair of spinors related to each other by time-reversal symmetry. The keyword "Relativistic zora spin" should be present to invoke calculations with spin-orbital coupling. The user also has to specify

```
Kspace 1
```

(Gamma-point calculation). The g-tensor is then computed from the HOMO spinor at the gamma point. In the output, the user can find two-contributions to g-tensor: one that stems from K\sigma operator and a second one, that stems from orbital angular momentum. By default, GIAO and spin-Zeeman corrections are not included, from our experience these corrections are quite small.

Electric Field Gradient (EFG)

```
EFG (block type)
```

The electronic charge density causes an electric field, and the gradient of this field couples with the nuclear quadrupole moment, that some (non-spherical) nuclei have and can be measured by several spectroscopic techniques. The EFG tensor is the second derivative of the Coulomb potential at the nuclei. For each atom it is a 3x3 symmetric and traceless matrix. Diagonalization of this matrix gives three eigenvalues, which are usually ordered by their decreasing absolute size and denoted as |Vzz|, |Vyy|, |Vxx|. . The result is summarized by the largest eigenvalue and the asymmetry parameter.

This options honors the `SelectedAtoms` key, in which case only the EFG will be calculated for the selected atoms.

NMR

```
NMR (block type)
SuperCell true/false
nmratom atom number
End
```

With the NMR option the shielding tensor is calculated. There are essentially two methods: the super cell method and the single-dipole method.

I) The super cell method is according to the implementation by Skachkov *et al.*[37]. The symmetry will automatically be set to NOSYM. The unit cell should not be chosen too small. Generating super cells can be conveniently done with the GUI.

II) The other method is the single-dipole method. In principle one can now use the primitive cell [50]. In practice also this method needs to be converged with super cell size. However, depending on the system the required super cell may be much smaller. At a given super cell size this method is more expensive than the super cell method.

`SuperCell`

This is the switch between the two methods, either the super cell, or the single-dipole method.

`nmratom`

Determines for which atom the shielding tensor is calculated. If you want more than one atom (or all) leave this line out. In that case the atoms for which to do NMR is controlled by the `SelectedAtoms` key. Here is an example how to select three atoms for NMR:

```
nmr
end

SelectedAtoms 3, 5, 9 ! leave this out to get NMR for ALL atoms
```

2.9 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).

Print key

`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.

`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.

`Fit`

Print more details about the fitting procedure, such as the fit coefficients at each SCF cycle.

`Symmetry`

Print the symmetry operators.

Thresholds

`POPTHRESHOLD`

Threshold for printing Mulliken population terms. Default 1e-2. Works with `print orbpop`.

`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 printed. Works with `print eigens`.

More options

The following print options only make sense when debugging the code.

`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
```

Debug key*

(programmers only)

`DEBUG`

The argument is usually a (list of) subroutine names that you want to debug. (The programmer can check for this in the same way as PRINT or ALLOW keys.)

2.10 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`.

Charges

Band prints always three charge analyses. Namely the Voronoi charges, the Mulliken analysis, and the Hirshfeld charges (see the ADF documentation on more details). The calculation comes at virtually no cost, so there are no keys associated with it. This is different for the Bader analysis.

Bader Analysis (AIM)

`GridBasedAIM (block-type)`

Invoke the ultra fast grid based Bader analysis [33,34].

```
GridBasedAIM
    { SmallDensity rhosmall }
    { Iterations n }
    { UseStartDensity true/false }

End
```

Where *rhosmall* (default 1e-6) is the value below which the density is ignored. This should not be chosen too small because it may lead to un-assignable grid points. The number *n* (default 40) determines the maximum number of steps that may be taken to find the nuclear attractor for a grid point. Finally, the analysis can be performed on the startup density rather than on the final density with the UseStartDenstiy sub key.

Also the critical points of the density can be determined. The algorithm starts from a regular mesh of points, and from each of these it walks towards its corresponding critical point.

```
AIMCriticalPoints
    { GridPadding pad }
    { GridSpacing space }
    { eqvPointsTol tol }
End
```

Where *pad* (default 0.7 bohr) determines how much extra space is added to the starting guess domain in the search for the critical points. The variable *space* (default 0.5 bohr) determines the distance between the initial trial points. Finally, *tol* (default 0.27 bohr) is used as a criterion to whether or not to critical points are the same.

Density of States

DOS

`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).

Gross populations

`GrossPopulations (block-type)`

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

```
GrossPopulations
   { iat lq }
   { FragFun jat 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` *jat ifun*, which refers to the function *ifun* of atom *jat*, 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
   EndSum
End
```

Overlap populations

`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
```

Band structure

The band structure is best examined with the GUI module "bandstructure". The band gap (if any) is printed in the output. Here is an example for the NaCl crystal

```
----------------------------------------
Band gap information
----------------------------------------
Number of valence electrons              16
Valence band index                        8
Top of valence band (a.u.)           -0.192
Bottom of conduction band (a.u.)     -0.039
Band gap (a.u.)                       0.153
Band gap (eV)                         4.173
Band gap (kcal)                      96.235
```

With the normal k-points the band structure looks rather coarse. The brute force solution is simply to crank up the KSPACE parameter, but this is very expensive and not needed for the precision. Band has an option to interpolate the band curves for the path through the Brillouin zone.

Band structure interpolation

```
BZStruct (block-type)
```

Influences how band outputs the band structure to the RUNKF file.

```
BZStruct
  Interpol ipol
  Enabled {true/false}
  Automatic {true/false}
End
```

`Enabled`: Whether or not to generate band structure data

`Interpol`: The higher the value of *ipol* the more refined the interpolation

`Automatic`: Whether or not to use the automatic path along high symmetry lines in the BZ

`BZPath (block-type)`

User defined path through the Brillouin zone.

```
BZPath
  kmesh mesh
  path
End
```

When `BZStruct%Automatic` is false you can specify manually a path. Increasing *mesh* leads to more points per line segment. The line segments are specified by some path sub keys. The coordinates are in terms of reciprocal lattice vectors. If you want to make a jump in the BZ, you need to specify a new path. Here is an example

```
bzpath
 kmesh 2
 path
  0.25 0.25 0.25
  0 0 0.5
  0 0 0
  0.5 -0.5 0.5
  0.25 0.25 0.25
  0 0 0
 subend
 path
  0 0 0.5
  0.5 -0.5 0.5
 subend
end
```

Effective mass

`EffectiveMass`

`EffectiveMass (block-type)`

In a semi conducter the mobility of the electrons and holes is related to the curvature of the bands at the top of the valence band and the bottom of the conduction band. With the effective mass option, this curvature is obtained with numerical differentiation. The estimation is done with the specified step size,

and twice the specified step size, and both results are printed to give a hint on the accuracy. By far the most convenient way to use this key is without specifying any subkeys.

```
EffectiveMass
  StepSize step
  NumAbove na
  NumBelow nb
  UniqueKPoints k1, k2, ...
End
```

`StepSize`: Size of the step taken in reciprocal space to perform the numerical differentiation.

`NumAbove`: Number of bands to take into account above the Fermi level.

`NumBelow`: Number of bands to take into account below the Fermi level.

`UniqueKPoints`: List of unique k-points where to print the effective mass. Omitting this, it will use the top of valence band and the bottom of the conduction band.

Properties at nuclei

`PropertiesAtNuclei`

A number of properties can be obtained near the nucleus. An average is taken over a tiny sphere around the nucleus. Tecnically the following properties are available.

```
PropertiesAtNuclei
  vxc[rho(fit)]
  rho(fit)
  rho(scf)
  v(coulomb/scf)
  rho(deformation/fit)
  rho(deformation/scf)
End
```

Physically *rho(scf)*, being the electronic density, is the most relevant one.

Form factors

`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

A fragment feature is available albeit rather primitive. It allows for the analysis of the DOS in a fragment basis and for the calculation of the deformation density with respect to fragment densities. 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. 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 in the next run you need to rename the result file ("RUNKF"), to something like "frag.runkf", see the example script discussed below.

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 overlayer with the substrate. With one or more `Fragment` keys you specify which fragment file(s) to use and to which atoms they should be mapped. It is allowed to have more than one fragment. The sub key `Labels` of a `Fragment` gives you the possibility to introduce labels for the fragment orbitals. Finally you can specify which fragments to use in the DOS analysis, via the `DosBas` key.

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 provided example is a slab calculation of Cu with a CO molecule adsorbed. A DOS analysis is performed in terms of the Cu atomic orbitals and the CO molecular orbitals. Note in the first step 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 is saved as CO.runkf, and is input for the second step of this example. Some of the orbital labels are adapted by specifying `Labels`. In the remaining steps this example demonstrates how to obtain the deformation density with respect to the sum of fragments (CO molecule + bare Cu slab) densities.

Fragment key

`Fragment (block-type)`

Define a fragment. 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. It has a subkey `Labels` so that you can assign meaningful names to the orbitals. You can define several fragments. Example

```
Fragment
  1 3 !  atom 1 of this fragment is assigned to third atom
  2 4 !  atom 2 of this fragment is assigned to fourth atom
  Labels
   Sigma
   Sigma*
   Pi_x
   Pi_y
   Pi_x*
   Pi_y*
  Subend
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.

2.11 Expert options*

Symmetry

The symmetry of the system is automatically detected. Normally the symmetry of the initial system is maintained. One can lower the symmetry with the Symmetry key. In such cases the keyword POTENTIALNOISE can force the solution away from the initial symmetry.

SYMMETRY

The most common option is NOSYM forcing the program to run without any symmetry.

It is also possible to run in a lower symmetry other than NOSYM but this is more involved (i.e. for experts only). The argument should be a list of numbers, representing the operators to maintain. The way to proceed is as follows. First run the calculation with the following added to your input

```
print symmetry
stopafter gemtry
```

and then you look in the output for (here the first four operators are listed)

```
    64    SYMMETRY OPERATORS:

  NO               MATRIX              TRANSL        AXIS    DET
ROTATION

------------------------------------------------------------------------

  1)        1.000   0.000   0.000       0.000       0.000    1.0
1
            0.000   1.000   0.000       0.000       0.000
            0.000   0.000   1.000       0.000       1.000

  2)        1.000   0.000   0.000       0.000       0.000    1.0
1
            0.000   1.000   0.000       5.400       0.000
            0.000   0.000   1.000       0.000       1.000

  3)        1.000   0.000   0.000       5.400       0.000    1.0
1
            0.000   1.000   0.000       0.000       0.000
            0.000   0.000   1.000       0.000       1.000

  4)        1.000   0.000   0.000       5.400       0.000    1.0
1
            0.000   1.000   0.000       5.400       0.000
            0.000   0.000   1.000       0.000       1.000
...
```

from this list you should select the desired operators and use that in your final calculation, for example

```
Symmetry 1 7 21 31
```

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.

Excited states

By default the levels are occupied according to the aufbau principle. In some cases it is possible to create holes below the fermi level with the `Occupations` key

`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
   irrepno occupations_alpha {// occupations_beta}
   ...
End
```

the *irrepno* 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.

The programmer key

The programmer has many options, most of which are set automatically. However, with the programmer key you can override many default behaviors. An overview is printed early in the output, and looks like

```
====================
 R U N    C O N F I G
 ====================


 Calculate gradient of basis  .  .  .  .  .  .  .  .  .  .  .  T
 Calculate sec. der. of basis .  .  .  .  .  .  .  .  .  .  .  F
 Calculate operator T working on basis .  .  .  .  .  .  .  .  T
 Calculate gradient of fit .  .  .  .  .  .  .  .  .  .  .  .  T
 Calculate sec. der. of fit.  .  .  .  .  .  .  .  .  .  .  .  F
 Calculate gradient of fit pot.  .  .  .  .  .  .  .  .  .  .  T
 Kin. energy via grad.  .  .  .  .  .  .  .  .  .  .  .  .  .  F
 Calculate gradient of energy .  .  .  .  .  .  .  .  .  .  .  T
 Calculate kin. energy density.  .  .  .  .  .  .  .  .  .  .  F
 Calculate nuc. grad of kin. energy density  .  .  .  .  .  .  F
 Calculate der. of density .  .  .  .  .  .  .  .  .  .  .  .  T
 Calculate second der. of density.  .  .  .  .  .  .  .  .  .  F
 Calculate nuc. grad. of density .  .  .  .  .  .  .  .  .  .  T
 Calculate nuc. grad. of grad. density .  .  .  .  .  .  .  .  F
 Exact rho during SCF.  .  .  .  .  .  .  .  .  .  .  .  .  .  T
 Exact grad. rho during SCF.  .  .  .  .  .  .  .  .  .  .  .  F
```

```
Exact grad. rho post SCF  . . . . . T
Store history of dens. matrix.  . . . . . F
Store original Bloch functions  . . . . . T
Direct  . . . . . F
Direct fit . . . . . F
Direct basis  . . . . . F
Calculate DOS . . . . . T
Calculate band structure  . . . . . T
Calculate EFG . . . . . F
```

Some of these option are controllable via the Programmer key.

`Programmer (block-type)`

`FlioSinglePrecision` ***{true**|false}*

The basis and fit functions are written and read to so-called FLIO files. With this option the double precision numbers can be converted to single precision, saving half of the IO. The results tend to be still fairly accurate. SCF convergence is usually possible to 1e-5. For deeper convergence disable this option.

`DirectBas` ***{true**|**false}***

Normally basis functions (and possibly derivatives) are calculated once and stored on disk. Enabling this option causes band not to store the basis functions but to recalculate them when needed. This reduces the amount of IO, at the cost of increased CPU.

`DirectFit` ***{true**|**false}***

Normally the fit functions, their Coulomb potentials (and possibly first and second derivatives) are calculated once and stored on disk. Enabling this option causes band not to store the first and derivatives but to recalculate them when needed. This reduces the amount of IO, at the cost of increased CPU. Note that the fitfunctions and potentials are currently always stored, so the fit is not yet completely direct.

`ExactGradRho` ***{true**|**false}***

The gradient (and second derivative) of the density are usually obtained via the fitted density. With this option you force the program to use the exact density gradient. This requires the calculation of basis set derivatives.

`KinViaGrad` ***{true**|**false}***

Matrix elements of the kinetic energy operator are normally calculated as

$T_{ij} = \int \psi_i \, 1/2 \, \nabla \cdot \nabla \, \psi_j$

Using partial integration this can be rewritten as

$T_{ij} = -\int (\nabla \psi_i) \, 1/2 \cdot \nabla \, \psi_j$

and requires the calculation of the gradient of the basis functions.

`UseSharedMemory` ***{true**|false}***

Whether or not to use shared memory in a band calculation. Doing so might reduce the memory requirement. A reason to disable it might be a bug (like a hangup or race condition) due to this feature.

`DidervCompat` ***{***true***|false}***

Originally all radial derivatives of NAOs were done numerically. However, more accurate derivatives are available from the DIRAC subroutine. To get the old (and slightly worse) derivatives set this option to *true*.

3 Recommendations, Problems, Questions

3.1 Recommendations

Performance for large unit cells

In the present version the standard settings are not yet optimized for large unit cells. Band usually errs on the conservative side. Here are some tips on how to get a much better performance. To be sure of the correctness of the tips, you should perform some tests.

On-the-fly calculation of basis and fit

Band has two ways to handle basis (and fit) functions. Either they are pre-calculated and cached on disk, or they are calculated directly on-the-fly when needed. Which choice is best depends strongly on the job and computer. If you see at the end of the output a big difference between the cpu time and the elapsed time, this is a clear indication that your hardware prefers the direct option for your job. Here is the end of an example output

```
 =================
 Timing Statistics
 =================

 Total Used :     CPU=   2349.05       System=      425.09      Elapsed=
4299.24
```

In this case you see that the elapsed time was roughly twice the cpu time. If you encounter this, or even a worse ratio, you should try the direct option, see `Programmer%DirectBas` and `Programmer%DirectFit`. For sure the CPU time will go up, but the elapsed time, the time that it takes to finish your job, might go down.

tails

The most important parameter is the `Tails` parameter. With this linear scaling is achieved for most parts of the program. By default it is 1e-6, which is a conservative value. According to our experience, total energies are still accurate using a setting like

```
Tails Bas=1e-3 Rosa
```

The extra key is named after a former student Rosa Bulo who found out that the largest error caused by the tails approximation comes from contracted functions. She suggested a pragmatical fix that makes makes the criterion more strict for contracted core-like functions.

confinement*

The localization of basis functions can be improved by confining the orbitals. See the option `AtomType%Confinement`, the only drawback is that is not user friendly to set up, because it does not work with the `BasisDefaults` key.

reciprocal space

Large unit cells have little dispersion, so you can get away with few k-points. Sometimes even a single k-point (`KSPACE 1`) might suffice.

Madelung summations: especially in bulk systems

When routines such as `ATOMIC` and `VMULTI` take a long time, visible in the log file, you can try to reduce the Madelung screening parameter. Band uses fit functions to calculate the Coulomb potential of the neutral and small deformation density. You can see how large Band has chosen this parameter in the output

```
 ======================================================================
1                            **  Bloch Summations  **



 Cut off criterion                                      0.100E-03
 Max. extension of the functions                       17.9095
 Max. cell distance taken                              37.7560
 Nr. of cells                                          19

 Fermi parameters for the Coulomb potential:
   Break off point                                     20.0000
   Fall off parameter                                   2.2361
```

This is example output for a slab. Here the Coulomb interaction screening parameter has been set to 20 bohr, leading to a reasonable number of cells. For large 3d cells you can get an incredible amount of cells, due to an unnecessary large choice of the screening parameter. See key `Screening%RMadel`.

reduced basis set

When starting work on a large unit cell it is wise to start with a DZ basis. With such a basis, any of the above suggestions can be tested. However, for most properties, the DZ basis is probably not very accurate. You can next go for the TZ2P basis set, but that may be a bit of overkill. One reasonable trick is to remove *f* polarization functions (if present) from this basis set. So for copper you can comment out the *4f* STO

```
AtomType Cu
DIRAC Cu
 7  5
VALENCE
1S
2S
2P
3S
3P
4S  1
3D
SubEnd
BasisFunctions
  3D     1.280000
  3D     6.900000
```

```
  4S     0.850000
  4S     2.450000
  4P     1.370000
!  4F     2.200000
SubEnd
FitFunctions
  1S    48.900000
  ...
```

Note that this will save you 7 functions per atom. If this is still unwieldy, you can also decide to start from the DZ basis and add a polarization function by hand.

Frozen core for *5d* elements

The standard basis sets TZ2P are not optimal for third-row transition elements. The V basis offers the possibility to freeze the 4f,5s, and 5p functions. With this you can usually still get quite good results. Sometimes you need to relax the frozen core dependency criterion

```
Dependency Core=0.8 ! The frozen core overlap may not be exactly 1
```

If you want to use the TZ2P you can change it like this one for Au

```
AtomType Au
DIRAC Au
14 12 ! note: now 12 core functions, was 10
VALENCE
1S
2S
2P
3S
3P
3D
4S
4P
4D
4F
5S
5P
5D    ! from here on valence
6S  1
SubEnd
BasisFunctions
!  5S     4.600000
  6S     3.150000
  6S     1.210000
!  5P     6.650000
!  5P     3.100000
  5D     5.050000
  5D     1.500000
  6P     1.950000
  5F     2.500000
SubEnd
...
```

Memory usage

Another issue that can important is the choice `CPVector` (say the vector length of you machine) and the number of k-points processed together during the calculation of the parameters. In the output you see the used value

```
                    =========================
                    = Numerical Integration =
                    =========================


 TOTAL NR. OF POINTS                                    4738
 BLOCK LENGTH                                            256
 NR. OF BLOCKS                                            20
 MAX. NR. OF SYMMETRY UNIQUE POINTS PER BLOCK             35
 NR. OF K-POINTS PROCESSED TOGETHER IN BASPNT              5
 NR. OF SYMMETRY OPERATORS (REAL SPACE)                   48
 SYMMETRY OPERATORS IN K-SPACE                            48
```

If you want to change the default settings you should specify *both* `CPVector` and `KGRPX`. The optimal combination depends on the calculation and on the machine and bigger is not necessarily better. Example

```
CPVector 1024
KGRPX 3
```

3.2 Trouble Shooting

SCF does not converge

Some systems are more difficult to converge than others. A Pd slab for instance is much more easy to converge than an Iron slab. Generally what you do in a problematic case is to go for more conservative settings. One of the first things to try is to use the Convergence%Degenerate key. The two main option are to decrease `SCF%Mixing` and/or `DIIS%Dimix`. If that all does not work you can try to use temperature smearingConvergence%ElectronicTemperature (but this only works for 2D systems). Add to you input

```
SCF
Mixing 0.05 ! more conservative mixing
End

Diis
DiMix 0.1 ! also more conservative strategy for DIIS procedure
End

Convergence
Degenerate Default ! For most calculations this is quite a good idea
anyway
ElectronicTemperature 0.03 0.01 0.003 ! Try to converge with
progressively lower temperatures
End
```

Also the `Programmer%FlioSinglePrecision` key can have effect. For some hard systems, using double precision significantly reduces the number of SCF iterations. However, it also can significantly increase the elapsed time.

You may also try to use global fitting, see `FitMethod%Inverse`.

Geometry does not converge

One thing that you should make sure that at least the SCF converges. If that is so, then maybe the gradients are not accurate enough. Here are some settings to improve the accuracy of the gradients

```
RadialDefaults
  NR 10000   ! more radial points
End

integration
accint 6  ! more integration points without affecting other tolerances
end

Programmer
fliosSinglePrecision false ! do double precision disk-IO.
end
```

If this does not help you could consider using a larger fit set, see `BasisDefaults%FitType`.

Negative frequencies phonon spectrum

When doing a phonon calculation one sometimes encounters unphysical negative frequencies. There are two likely causes: either the geometry was not in the minimum geometry (see `GeoOpt%Converge`), or the step size used in the Phonon run is too large (see `PhononConfig%StepSize`). Also general accuracy issues may be the cause, such as numercial integration, k-space integration and fit error.

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 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. There are two ways out: using confinement or removing basis functions.

Using confinement

Usually the dependency problem is due to the diffuse basis functions. This is especially so for highly coordinated atoms. One way to reduce the range of the functions is to use the `Confinement` key. In a slab you could consider to use confinement only in the inner layers, and to use the normal basis to the surface layers. The idea is that basis functions of the surface atoms can describe the decay into the vacuum properly, and that inside the slab the diffuseness of the functions is not needed. If all the atoms of the slab are of the same type, you should make a special type for the inner layers: simply put them in a separate `Atoms` block. The confinement can be specified per type.

Removing basis functions

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.

Frozen core too large

Band calculates the overlap matrix of the core functions, and this should approximate the unit matrix. When the deviation is larger then the frozen-core overlap criterion the program stops. The default criterion (0.98) is fairly strict. The safest solution is to choose a smaller frozen core. For performance reasons, however, this may not be the preferred option. In practice you might still get reliable results by setting the criterion to 0.8, see the `Dependency` keyword. For the *5d* transition metals, for instance, you can often freeze the 4f orbital, thus reducing the basis set considerably. We strongly advise you to compare these results to a calculation with a smaller core. Such tests can be performed with a smaller unit cell or with a smaller KSPACE parameter.

Unstable fit

Sometimes the fit can become unstable, usually because it is too dependent. Some of the symptoms are: a charge error during the SCF, a dependent fit set, or a large fit error as printed in the energy section of the output. Usually the stability is improved by removing some diffuse fit functions. Another thing that might help is to use the more expensive ORTHOGONALFIT option.

3.3 Various issues

Breaking the symmetry

In some cases you want to break the symmetry. An example of this is when you want to get the anti-ferromagnetic state of Fe. Another common example is when you want to apply geometry constraints on atoms.

The easiest way to do this is of course to disable all symmetry, see `SYMMETRY%NOSYM`, but this might make your calculation more expensive than is needed. A bit more elegant way is to define separate types for the equivalent atoms. Here follows example input for antiferrmomagnetic iron

```
ATOMS Fe
0.0 0.0 0.0
end

! second iron as new type to break the symmetry
atoms Fe
-1.435 -1.435 1.435
END

Lattice
 -1.435 1.435 1.435
 1.435 -1.435 1.435
 2.87 2.87 -2.87
```

```
End

CONVERGENCE
CRITERION 1.0e-4
Degenerate default
SpinFlip 2  ! Flip (startup) spin density at second atom
END
```

Another solution is to use the expert `SYMMETRY` keyword.

Labels for the basis functions

You see the labels for the basis functions in for instance the DOS section of the output. The labels are also used in combination with options like `Print Eigens` and `Print OrbPop`. (See also `Print Orbitallabels`).

What do the labels look like? 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 will just get simple numbers as labels:

CORE STATE 1
CORE STATE 2

With the fragment option you can give meaningful names to the fragment option, see `Fragment%Labels` and `DosBas`.

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-*un*restricted 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.

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.

4 References

1. S.H. Vosko, L. Wilk and M. Nusair, *Accurate spin-dependent electron liquid correlation energies for local spin density calculations: a critical analysis.* Canadian Journal of Physics **58**, 1200 (1980)

2. H. Stoll, C.M.E. Pavlidou and H. Preuss, *On the calculation of correlation energies in the spin-density functional formalism.* Theoretica Chimica Acta **49**, 143 (1978)

3. A.D. Becke, *Density-functional exchange-energy approximation with correct asymptotic behavior.* Physical Review A **38**, 3098 (1988)

4. J.P. Perdew and Y. Wang, *Accurate and simple density functional for the electronic exchange energy: generalized gradient approximation.* Physical Review B **33**, 8800 (1986)

5. J.P. Perdew, J. A. Chevary, S. H. Vosko, K. A. Jackson, M. R. Pederson, D. J. Singh and C. Fiolhais, *Atoms, molecules, solids, and surfaces: Applications of the generalized gradient approximation for exchange and correlation.* Physical Review B **46**, 6671 (1992)

6. J.P. Perdew, *Density-functional approximation for the correlation energy of the inhomogeneous electron gas.* Physical Review B **33**, 8822 (1986)

7. C. Lee, W. Yang and R.G. Parr, *Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density.* Physical Review B **37**, 785 (1988)

8. B.G. Johnson, P.M.W. Gill and J.A. Pople, *The performance of a family of density functional methods.* Journal of Chemical Physics **98**, 5612 (1993)

9. T.V. Russo, R.L. Martin and P.J. Hay, *Density Functional calculations on first-row transition metals.* Journal of Chemical Physics **101**, 7729 (1994)

10. R. van Leeuwen and E.J. Baerends, *Exchange-correlation potential with correct asymptotic behavior.* Physical Review A **49**, 2421 (1994)

11. R. Neumann, R.H. Nobes and N.C. Handy, *Exchange functionals and potentials.* Molecular Physics **87**, 1 (1996)

12. J.P. Perdew, K. Burke and M. Ernzerhof, *Generalized Gradient Approximation Made Simple.* Physical Review Letters **77**, 3865 (1996)

13. B. Hammer, L.B. Hansen, and J.K. Norskøv, *Improved adsorption energetics within density-functional theory using revised Perdew-Burke-Ernzerhof functionals.* Physical Review B **59**, 7413 (1999)

14. J.P. Perdew, A. Ruzsinszky, G.I. Csonka, O.A. Vydrov, G.E. Scuseria, L.A. Constantin, X. Zhou and K. Burke, *Restoring the Density-Gradient Expansion for Exchange in Solids and Surfaces.* Physical Review Letters **100**, 136406 (2008)

15. J. Tao, J.P. Perdew, V.N. Staroverov and G.E. Scuseria, *Climbing the Density Functional Ladder: Nonempirical Meta-Generalized Gradient Approximation Designed for Molecules and Solids.* Physical Review Letters **91**, 146401

16. Y. Zhao, D.G. Truhlar, *A new local density functional for main-group thermochemistry, transition metal bonding, thermochemical kinetics, and noncovalent interactions.* Journal of Chemical Physics **125**, 194101

17. P.H.T. Philipsen, E. van Lenthe, J.G. Snijders and E.J. Baerends, *Relativistic calculations on the adsorption of CO on the (111) surfaces of Ni, Pd, and Pt within the zeroth-order regular approximation.* Physical Review B **56**, 13556 (1997)

18. P.H.T. Philipsen, and E.J. Baerends, *Relativistic calculations to assess the ability of the generalized gradient approximation to reproduce trends in cohesive properties of solids.* Physical Review B **61**, 1773 (2000)

19. E.S. Kadantsev, R. Klooster. P.L. de Boeij and T. Ziegler, *The Formulation and Implementation of Analytic Energy Gradients for Periodic Density Functional Calculations with STO/NAO Bloch Basis Set.* Molecular Physics **105**, 2583 (2007)

20. E.S. Kadantsev and T. Ziegler, *Implementation of a Density Functional Theory-Based Method for the Calculation of the Hyperfine A-tensor in Periodic Systems with the Use of Numerical and Slater Type Atomic Orbitals: Application to Paramagnetic Defects.* Journal of Physical Chemistry A **112**, 4521 (2008)

21. E.S. Kadantsev and T. Ziegler, *Implementation of a DFT Based Method for the Calculation of Zeeman g-tensor in Periodic Systems with the use of Numerical and Slater Type Atomic Orbitals.* Journal of Physical Chemistry A **113**, 1327 (2009)

22. F. Kootstra, P.L. de Boeij and J.G. Snijders, *Efficient real-space approach to time-dependent density functional theory for the dielectric response of nonmetallic crystals.* Journal of Chemical Physics **112**, 6517 (2000)

23. P. Romaniello and P.L. de Boeij, *Time-dependent current-density-functional theory for the metallic response of solids.* Physical Review B **71**, 155108 (2005)

24. J.A. Berger, P.L. de Boeij and R. van Leeuwen, *Analysis of the viscoelastic coefficients in the Vignale-Kohn functional: The cases of one- and three-dimensional polyacetylene.* , Physical Review B **71**, 155104 (2005)

25. P. Romaniello and P.L. de Boeij, *Relativistic two-component formulation of time-dependent current-density functional theory: application to the linear response of solids.* , Journal of Chemical Physics **127**, 174111 (2007)

26. J.P. Perdew, A. Ruzsinszky, G. I. Csonka, L. A. Constantin, and J. Sun, *Workhorse Semilocal Density Functional for Condensed Matter Physics and Quantum Chemistry.* , Physical Review Letters **103**, 026403 (2009)

27. C. Adamo and V. Barone, *Exchange functionals with improved long-range behavior and adiabatic connection methods without adjustable parameters: The mPW and mPW1PW models.* Journal of Chemical Physics **108**, 664 (1998)

28. Y. Zhang and W. Yang, *Comment on "Generalized Gradient Approximation Made Simple".* Physical Review Letters **80**, 890 (1998)

29. C. Adamo and V. Barone, *Physically motivated density functionals with improved performances: The modified Perdew.Burke.Ernzerhof model.* Journal of Chemical Physics 1996 **116**, 5933 (1996)

30. N.C. Handy and A.J. Cohen, *Left-right correlation energy.* Molecular Physics **99**, 403 (2001)

31. M. Swart, A.W. Ehlers and K. Lammertsma, *Performance of the OPBE exchange-correlation functional.* Molecular Physics 2004 **102**, 2467 (2004)

32. S. Grimme, *Semiempirical GGA-Type Density Functional Constructed with a Long-Range Dispersion Correction.* Journal of Computational Chemistry **27**, 1787 (2006)

33. J.I. Rodríguez, A.M. Köster, P.W. Ayers, A. Santos-Valle, A. Vela and G. Merino, *An efficient grid-based scheme to compute QTAIM atomic properties without explicit calculation of zero-flux surfaces.* Journal of Compututational Chemistry **30**, 1082 (2009)

34. J.I. Rodríguez, R.F.W. Bader, P.W. Ayers, C. Michel, A.W. Götz and C. Bo, *A high performance grid-based algorithm for computing QTAIM properties.* Chemical Physics Letters **472**, 149 (2009)

35. J. Tersoff and D. R. Hamann, *Theory of the scanning tunneling microscope.* Physical Review B **31**, 505 (1985)

36. A. Klamt and G. Schüürmann, *COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient.* Journal of the Chemical Society: Perkin Transactions **2**, 799 (1993)

37. D. Skachkov, M. Krykunov, E. Kadantsev, and T. Ziegler, *The Calculation of NMR Chemical Shifts in Periodic Systems Based on Gauge Including Atomic Orbitals and Density Functional Theory.* Journal of Chemical Theory and Computation **6**, 1650 (2010)

38. J.L. Pascual-ahuir, E. Silla and I. Tuñon, *GEPOL: An improved description of molecular surfaces. III. A new algorithm for the computation of a solvent-excluding surface.* Journal of Computational Chemistry **15**, 1127 (1994)

39. A. Klamt and G. Schüürmann, *COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient.* Journal of the Chemical Society: Perkin Transactions **2**, 799 (1993)

40. B. Delley, *The conductor-like screening model for polymers and surfaces.* Molecular Simulation **32**, 117 (2006)

41. N.L. Allinger, X. Zhou, J. Bergsma, *Molecular mechanics parameters*, Journal of Molecular Structure: THEOCHEM **312**, 69 (1994)

42. S. Grimme, J. Anthony, S. Ehrlich, and H. Krieg, *A consistent and accurate* ab initio *parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu*, The Journal of Chemical Physics **132**, 154104 (2010).

43. S. N. Steinmann, and C. Corminboeuf, *Comprehensive Benchmarking of a Density-Dependent Dispersion Correction*, Journal of Theory and Computation **132**, 3567 (2011).

44. S. Grimme, S. Ehrlich, and L. Goerigk, *Effect of the Damping Function in Dispersion Corrected Density Functional Theory*, Journal of Computational Chemistry**32**, 1457 (2011).

45. Ph. Haas, F. Tran, P. Blaha, and K. H. Schwartz, *Construction of an optimal GGA functional for molecules and solids*, Physical Review B**83**, 205117 (2011).

46. F. Tran, and P. Blaha, *Accurate Band Gaps of Semiconductors and Insulators with a Semilocal Exchange-Correlation Potential*, Physical Review Letters**102**, 226401 (2009).

47. D. Alfè, *PHON: A program to calculate phonons using the small displacement method*, Computer Physics Communications**180**, 2622 (2009).

48. V.I. Asimov, F. Aryasetiawan, and A.I. Lichtenstein, *First-principles calculations of the electronic structure and spectra of strongly correlated systems: the LDA + U method*, Journal Physics: Condensed Matter **9**, 767 (1997).

49. M. Cococcioni, and S. de Gironcoli, *Linear response approach to the calculation of the effective interaction parameters in the LDA+U method*, Physical Review B**71**, 035105 (2005).

50. D. Skachkov, M. Krykunov, and T. Ziegler, *An improved scheme for the calculation of NMR chemical shifts in periodic systems based on gauge including atomic orbitals and density functional theory*, Canadian Journal of Chemistry **89**, 1150 (2011).

51. M. Kuisma, J. Ojanen, J Enkovaara, and T. T. Rantala, *Kohn-Sham potential with discontinuity for band gap materials*, Physical review **B82**, 115106 (2010).

52. L. Visscher, and K. G. Dyall, *DIRAC-FOCK ATOMIC ELECTRONIC STRUCTURE CALCULATIONS USING DIFFERENT NUCLEAR CHARGE DISTRIBUTIONS*, Atomic Data and Nuclear Data Tables **67**, 207 (1997).

53. A.D. Becke, *A multicenter numerical integration scheme for polyatomic molecules*, Journal of Chemical Physics **88**, 2547 (1988)

Keywords

Index