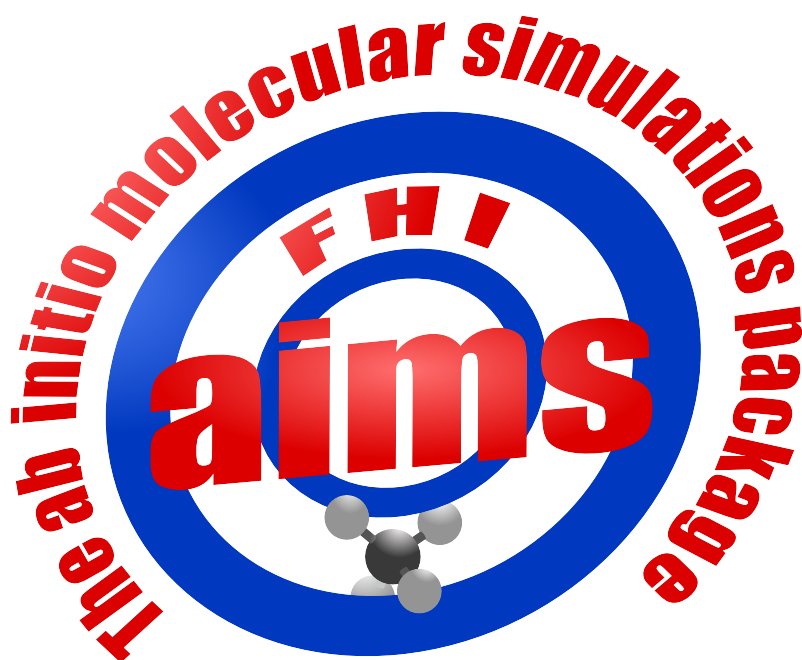


---

**Fritz Haber Institute**  
*ab initio* molecular simulations:  
**FHI-aims**

---



All-Electron Electronic Structure Theory  
with Numeric Atom-Centered Basis Functions

A Users' Guide

---

FHI-aims team  
Fritz-Haber-Institut der Max-Planck-Gesellschaft, Berlin  
and many contributors around the world.  
August 21, 2012

# Contents

<b>Introduction</b>	<b>5</b>
<b>1 Getting started with FHI-aims</b>	<b>7</b>
1.1 First step: Installation	7
1.2 Building the code	8
1.3 Running FHI-aims	10
1.4 Compilation options beyond the standard Makefile	13
<b>2 Input Files: Basic Handling</b>	<b>14</b>
2.1 The mandatory input files: <code>control.in</code> and <code>geometry.in</code>	15
2.2 Defaults for chemical elements: <code>species_defaults</code>	18
2.3 A very quick guide to ensuring numerical convergence with FHI-aims	20
2.3.1 Basis set	20
2.3.2 Hartree potential	22
2.3.3 Integration grid	23
2.4 Stopping a run: Files <code>abort_scf</code> and <code>abort_opt</code>	26
<b>3 The Full Monty: All Keywords and Capabilities</b>	<b>27</b>
3.1 Usability (convenience)	28
3.2 Physical model: Geometry, charge, spin, etc.	30
3.3 Electronic structure: Exchange, correlation (incl. DFT+U), and excited states	34
3.4 Specifying the basis (functions, empty sites, k-points, ...)	45
3.5 Integration, grids, and partitioning	57
3.6 Electron density update	67
3.7 Electrostatic (Hartree) potential	69

---

3.7.1	Non-periodic Ewald method . . . . .	70
3.8	Kinetic energy, scalar relativity, and spin-orbit coupling . . . . .	79
3.9	Eigenvalue solver and (fractional) occupation numbers . . . . .	85
3.10	SCF Cycle: Initialization, density mixing, preconditioning, convergence . . . . .	94
3.11	Energy derivatives (forces, stress) and geometry optimization . . . . .	110
3.12	Molecular dynamics . . . . .	122
3.13	Thermodynamic Integration . . . . .	132
3.14	Electronic constraints . . . . .	136
3.15	Embedding in external fields . . . . .	141
3.16	QM/MM Embedding . . . . .	144
3.17	$C_6/R^6$ corrections for long-range van der Waals (London dispersion) interactions . . . . .	148
3.18	Calculating nonlocal correlation energy within density functional approach . . . . .	151
3.18.1	Monte Carlo integration based vdW-DF . . . . .	151
3.18.2	Analytic integration scheme for non-selfconsistent and self-consistent vdW-DF . . . . .	156
3.19	Hartree-Fock, hybrid functionals, <i>GW</i> , <i>et al.</i> : All the details . . . . .	158
3.20	Periodic Hartree-Fock and hybrid functionals . . . . .	170
3.21	Large-scale, massively parallel: Memory use, sparsity, communication, etc. . . . .	174
3.22	Output options . . . . .	181
3.23	Deprecated keywords . . . . .	200
<b>4</b>	<b>Running FHI-aims: Guides to specific tasks</b> . . . . .	<b>207</b>
4.1	Ground state DFT: Total energies and relaxation . . . . .	208
4.2	Heavy elements ( $Z \gtrsim 30$ ): Modifications for scalar relativity . . . . .	214
4.3	k-point sampling in the Brillouin zone for semiconductors . . . . .	217
4.4	Plotting the band structure and density of states . . . . .	221
4.5	Visualizing charge densities and orbitals . . . . .	224
4.6	Calculation of vibrational and phonon frequencies . . . . .	227
4.7	Transition state search: Nudged Elastic Band method . . . . .	237
4.7.1	Theory and methods . . . . .	237
4.7.2	Usage . . . . .	237
4.8	Plugin for free-energy calculations with molecular dynamics: PLUMED . . . . .	242

---

4.8.1	Usage	242
4.9	Script based parallel tempering (a.k.a. replica exchange)	244
4.9.1	Usage	244
4.9.2	Output	246
4.10	Formation energies of charged defects	247
<b>5</b>	<b>The AITRANSS package</b>	<b>249</b>
5.1	Source code and supporting materials	250
5.2	Compiling the AITRANSS module	250
5.3	How to set-up and run transport calculations	251
5.3.1	FHI-aims run: input and output	251
5.3.2	What to be aware of before running AITRANSS module	251
5.3.3	How to create a mandatory file <code>tcontrol</code>	253
5.3.4	How to submit a transport calculation and its output	255
5.4	Keywords of file <code>tcontrol</code>	256
<b>A</b>	<b>Trouble-shooting</b>	<b>261</b>
A.1	Format flags required by some compilers	261
A.2	FHI-aims aborts with a segfault at the beginning of the first test run.	262
A.3	Use of FHI-aims with multithreaded BLAS (e.g., Intel's mkl)	263
A.4	Parallel runs across different file systems	263
A.5	Nearly singular basis sets: Strange results from small-unit-cell periodic calculation with many k-points	264
A.6	No convergence of the s.c.f. cycle even after many iterations	265
<b>B</b>	<b>Structure of the code</b>	<b>267</b>
B.1	Flow of the program	267
B.2	Commenting and style requests	271
	<b>Bibliography</b>	<b>278</b>
	<b>Index</b>	<b>278</b>

# Introduction

FHI-aims (“Fritz Haber Institute *ab initio* molecular simulations”) is a computer program package for computational materials science based only on quantum-mechanical first principles. The main production method is density functional theory (DFT) [51, 59, 25] to compute the total energy and derived quantities of molecular or solid condensed matter in its electronic ground state. In addition, FHI-aims allows to describe electronic single-quasiparticle excitations in molecules using different self-energy formalisms (*GW* and MP2), and wave-function based molecular total energy calculation based on Hartree-Fock and many-body perturbation theory (MP2, RPA, SOSEX, or the more encompassing renormalized second-order perturbation theory, RPT2).

The basic physical algorithms in FHI-aims concerning ground state DFT and applications are described in

Volker Blum, Ralf Gehrke, Felix Hanke, Paula Havu, Ville Havu, Xinguo Ren, Karsten Reuter, and Matthias Scheffler, *Computer Physics Communications* **180**, 2175-2196 (2009).

A copy of this paper can also be obtained from our web site:

<http://www.fhi-berlin.mpg.de/aims/> .

Please cite this reference if you use FHI-aims.

When making use of / reference to scalability, please refer to and cite

Ville Havu, Volker Blum, Paula Havu, and Matthias Scheffler, *Journal of Computational Physics* **228**, 8367-8379 (2009).

and also to the large-scale eigenvalue solver ELPA:

T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, and P. R. Willems, *Parallel Computing* **37**, 783-794 (2011).

Any application making use of functionality beyond LDA, GGA, or mGGA – i.e., Hartree-Fock, hybrid functionals, MP2, RPA, *GW*, etc. – should please refer to and cite

Xinguo Ren, Patrick Rinke, Volker Blum, Jürgen Wieferink, Alex Tkatchenko, Andrea Sanfilippo, Karsten Reuter, and Matthias Scheffler, *New Journal of Physics* **14**, 053020 (2012).

In the present documentation, we do not repeat these basic physical algorithms; rather, the focus is on the actual *use* of the methods in FHI-aims for a given task, including a full description of all input and output possibilities.

The rest of this document is organized as follows:

- In Chapter 1, a “quickstart” description attempts to give you all the necessary (but not more) information to get FHI-aims up and running on your own computer system, up to the first test run.
- Chapter 2 explains the basic input files and input philosophy very briefly. Some important remarks on choosing the numerical accuracy are summarized here.
- Chapter 3 gets into the gory details, summarizing *all* available input keywords and their meaning, sorted roughly by their expected use.
- A large chapter 4 is dedicated to some frequently required “meta-tasks” of electronic structure theory: Not just setting up a specific set of input files for a given run, but actually extracting some of the frequently required information from those runs. For the more complex tasks (e.g., a transition state search), we attempt to provide scripts that perform a series of well-defined runs automatically, the use of an external visualization tool, etc.
- In chapter 5 we provide a description of the AITRANSS (*ab initio* transport simulations) package which is a project under continuous development at the Institute of Nanotechnology of the Karlsruhe Institute of Technology (KIT), Germany, since 2002. When combined with FHI-aims, AITRANSS provides a post-processor module that enables calculation of the electron transport characteristics of molecular junctions based on a Landauer formalism in a Green’s function formulation.
- In the appendices, we suggest further reading, and also address frequently encountered issues (“troubleshooting”) that are either beyond our control (operating-system related issues come to mind), or simply require some level of experience to address.

In any case, we hope that this layout will be helpful for your specific purposes. We welcome feedback, in particular regarding issues from production settings that we might not yet have thought of / experienced ourselves. In any event: Happy computing with FHI-aims!

# Chapter 1

## Getting started with FHI-aims

### 1.1 First step: Installation

FHI-aims comes as a gzipped tar archive that can be extracted in any directory of your choice, e.g., by typing

```
gzip -d fhi-aims.tar.gz
tar -xvf fhi-aims.tar
```

at the command line of any current Unix-like system.

Before you ask: FHI-aims is designed to run on any current Unix-based or Unix-like system, such as Linux, IBM's aix, Sun's Solaris, or even Mac OS X. However, we do *not* support FHI-aims on Windows at this point. It is certainly possible to make it run on a Windows platform using the appropriate tools, but not simply out-of-the-box.

The full package then extracts itself into a directory `./fhi-aims`, with the following sub-directories:

- `bin/` : Location for any FHI-aims binaries built using the standard Makefile
- `doc/` : Contains possible further documentation.
- `species_defaults/` : Grids, basis sets and other defaults for chemical elements 1-102. These can be copy-pasted as "species" into the FHI-aims input file `control.in`. FHI-aims provides four levels of species defaults: "light", "tight", "really\_tight", and a partial set "light\_194" (see Sec. 2.2).
- `src/` : This directory, and its subdirectories, contain all FHI-aims source code files, as well as the Makefile needed to build this code.
- `testrun/` : Simple testruns to test and illustrate the basic functioning of the code. The input files provided here may also be used as templates for any new electronic structure calculations, rather than assembling them from scratch.

- *utilities/* : Some simple scripts to extract basic information from the standard output of FHI-aims: Visualization of geometries using the .xyz format, extracting a series of geometries during relaxation as a movie, or extracting the development of energies and forces during relaxation. There is also some more sophisticated infrastructure here: Script-based ab initio replica exchange molecular dynamics (Luca Ghiringhelli) and a basin-hopping framework to predict the structure of small clusters from scratch (Ralf Gehrke).

A README file in that directory contains some of the quickstart information given here in condensed format.

## 1.2 Building the code

*To state this at the outset: The “aimsclub” Wiki is the appropriate place to look for detailed compiler settings for specific platforms. Do not be shy to add your own, and do not be shy to ask in the Forum when in doubt. The information given in this chapter is essential as it explains the process, but the platform specific remarks in the Wiki may help you save some time.*

Since FHI-aims is distributed in source code form, the first task is to compile an executable program. For this, the following *mandatory prerequisites* are needed:

- A working Fortran 95 (or later) compiler. A good example for x86 type computers is Intel’s ifort compiler. A free but significantly slower compiler for all platforms is gfortran from the GNU compiler collection (<http://gcc.gnu.org/fortran>) or the g95 compiler (<http://www.g95.org>). Do not underestimate this slowdown, though – a factor of three or so is possible.
- A compiled version of the lapack library, and a library providing optimized basic linear algebra subroutines (BLAS). Standard commercial libraries such as Intel’s mkl or IBM’s essl provide both lapack and BLAS support. lapack can also be found at <http://www.netlib.org/lapack/>.  
*Having an optimized BLAS library for YOUR specific computer system(s) is critical for the performance of FHI-aims.* Very good free implementations include ATLAS (<http://math-atlas.sourceforge.net/>) or the Goto BLAS (<http://www.tacc.utexas.edu/resources/software/>).

You should also have a version of the Gnu make command, as not all ancient Unix make commands support our Makefile. Typically, Gnu make will already be present on your system, either as make, or possibly as gmake.

The next two prerequisites are *optional*, but absolutely encouraged for any advanced use of FHI-aims: Support for parallel architectures, and (separately) support for fully parallel Scalapack-based eigenvalue solvers. In these cases, you will also need:

- A version of MPI libraries for parallel execution, often already present on a parallel system (if not, <http://www.open-mpi.org/> provides one of several free implementations)

- Compiled versions of the scalapack library, and basic linear algebra *communication* subroutines (BLACS). Capable implementations can be found at <http://www.netlib.org/>, but are often provided already in the numerical libraries of many vendors (e.g., Intel MKL on Linux).

**The creation of a complete, MPI- and scalapack-enabled binary is effort well spent and should be the goal when compiling FHI-aims for any production purposes.**

The “aimsclub” Wiki gives compilation hints for many platforms (including MPI, BLACS and Scalapack, which we strongly recommend on any platform with more than one CPU). Please help by adding your own working settings there, and ask us if there are questions.

Once all prerequisites are in place, change directory to the *src/* directory, and open the Makefile in a text editor.

*You must adjust at least some system-specific portions of the Makefile—simply typing “make” and hoping that the problem will go away will not work.*

Usually, all you will have to do is to decide on one of the preconfigured make targets – “serial”, “mpi”, or “scalapack.mpi”. Near the top of the Makefile, a number of *mandatory* settings are commented for each target. Uncomment *only* the block of settings relevant to your chosen make target, and fill in the correct values of each variable (FC, FFLAGS, LAPACKBLAS, ...) for your computer system. *Note* that the Makefile itself contains detailed instructions and explanations regarding the meaning of these variables. Often, simply adjusting the compiler name, the location of your libraries (LAPACKBLAS, possibly MPIFC or SCALAPACK) will be sufficient. In addition, we *strongly* recommend that you consult the documentation of your compiler, in order to find out which optimization options *beyond* the generic “-O3” optimization level suggested preset in the Makefile will make a difference on your computer.

Finally, this brings us to the key step of the build process: Building the code. After the Makefile is adjusted, type

```
make <target>
```

at the command line, where “<target>” should be replaced by the target of your choice: “serial”, “mpi”, or “scalapack.mpi”.<sup>1</sup> If successful, this should build the desired FHI-aims binary (the compilation will take a while) and place it in the *bin/* directory mentioned above.

Building FHI-aims can take a while nowadays. If you have more than one processor on the machine for building FHI-aims, try

```
make -j <number_of_processors> <target>
```

---

<sup>1</sup>There is an additional target, `parser`, which builds an executable that stops after parsing the input. This binary can be used to check the validity of input files. (The `dry_run` keyword achieves the almost same effect with the full binary.)

This choice should speed up the process greatly.

You may also wish to keep your own copy of the Makefile, for instance to be able to work directly with the FHI-aims git repository without overwriting the general Makefile. In that case, just copy the standard Makefile to something like `Makefile.myname`, and use

```
make -j <number_of_processors> -f Makefile.myname <target>
```

Finally, we do note that there is some support for more sophisticated tasks, such as cross-platform builds (building binaries for different architectures from the same home directory) among the non-standard environment variables in the Makefile (see there).

## 1.3 Running FHI-aims

As a simple test run to establish the correct functioning of FHI-aims and also to familiarize yourself with the basic structure of the input and output files, we suggest you change directories to the `testruns/H2O-relaxation/` directory. The test run provided there relaxes a simple H<sub>2</sub>O molecule from an initial (distorted) structure to the stable one, and computes total energies, eigenvalues etc. along the way. Notice that the key convergence settings (basis sets and grids) in this example are chosen to be fast. The results (particularly the relaxed geometry) are still trustworthy, but we encourage you already here to explore more stringent convergence settings later. In fact, *always* explore the impact of critical convergence settings on the accuracy of key results in your own project.

In the `testruns/H2O-relaxation/` directory, type

```
../../bin/aims.version.serial.x < /dev/null | tee H20_test.own
```

at the command line. For “*version*”, you must insert the code version stamp that was actually downloaded and built (for example, 010110, 071711\_5, 081912, or whichever code version you are building).<sup>2</sup> Similarly, for make targets other than “*serial*”, you should use the appropriate binary including the necessary mpi command instead. On many (but not all) platforms, that command will be `mpirun`, and will also require you to specify the number of processors to be used by a flag.

The result will be an output stream on your computer screen (created by “*tee*”) which is also captured in an output file `H20_test.own`. Any critical information regarding computational settings, results (total energies, forces, geometries, ...), errors etc. should be contained in this file, which we encourage you to look at (yes, it is *meant* to contain human-readable and useful explanations). Any *other* output files are only written if requested, and will be covered in the later sections of this text.

---

<sup>2</sup>The FHI-aims version stamp can be modified to whatever you wish in `Makefile.backend` in the `src/` directory.

*The standard output stream or file contains any and all output that FHI-aims writes by default. For later use, you must save this output stream to disk in some way, using standard Unix redirections such as the `tee` command above or a simple redirect.*

Apart from the first expression given above, such redirections might look like this:

```
../../bin/aims.version.serial.x < /dev/null > H2O_test.own
```

or even like this:

```
nohup ../../bin/aims.version.serial.x < /dev/null > H2O_test.own 2>&1 &
```

The latter version decouples the FHI-aims run completely from your current login, and additionally saves any system error messages to the standard output file as well. With the above command sequence, you may safely log out from the computer in question, the code should keep running in the background. Take care to monitor your running processes using the `ps` Unix command. For instance, it is highly inadvisable to run ten instances of FHI-aims at once in the background on a single CPU and expect any reasonable performance of the computer at all. The above hints are just examples of general Unix command-line sequences. For a complete treatment, we recommend that beginners read a separate Unix textbook, or—often feasible—learn by doing and Google.

If successful (otherwise, consider the warnings three paragraphs below), you may wish to compare your results to those contained in our own output from this run, which is contained in the file `H2O.reference.out`. You should obtain exactly the same total energies, forces, and geometries as given in this file. Any information regarding timing is, of course, specific to your computer environment, and not necessarily the same.

The directory `testruns/H2O-relaxation/` contains two more files, `control.in` and `geometry.in`. These are the sole two input files required by FHI-aims, and are the most important files to learn about in the rest of this documentation. In brief, `geometry.in` contains any information related directly to a system's geometry – normally, this will be atomic positions (in Å) and perhaps lattice vectors for periodic calculations, but no more. Any other, method-related, input information is part of `control.in`.

In practice, we attempt to strike a balance between the information *needed* by `control.in`, and information set to safe defaults unless specified explicitly. For example, you *must* specify the level of theory (e.g., the fact that PBE exchange-correlation is used) and also the basis set employed. While it is highly useful to have this relevant information openly accessible, this would also create the need to personally edit a large amount of input before ever tackling the first run. For any information tied to the actual element (or “species”; arguably the most complex information required), we therefore provide ready-made template files for all elements (1-102) in the `species_defaults` directory. They are ready for copy-paste into `control.in`. These files will still benefit from some adjustment to your personal needs (for instance, the provided integration grids are set rather on the safe side, at the expense of more CPU time), but should greatly simplify the task.

Two final, important warnings regarding the execution of FHI-aims that are beyond our direct control:

- FHI-aims *requires* that the execution stack size available to you be large enough for some initial internal operations. Spare us the details (ample explanation of the meaning of the “stack” in Unix can be found elsewhere), but for reasons unbeknownst to us, some vendors limit the default user stack size to  $\approx 5$  MB in a time when typical available RAM per processor is 2 GB or more. If too little stack is available, your FHI-aims run will *segfault* shortly after the command was launched. In that case, type:

```
ulimit -s unlimited
```

(when using the bash shell or similar), or

```
limit stacksize unlimited
```

(when using the tcsh or similar).

```
echo $SHELL
```

will tell you which shell you are using. Ideally, this same setting should be specified in your `.profile`, `.bashrc`, or `.cshrc` login profiles. If “unlimited” does not work, try setting a large value instead, e.g., `ulimit -s 500000`.

- An important system settings for parallel execution is the environment variable

```
export OMP_NUM_THREADS=1
```

(the syntax is correct for the bash shell). When using Intel’s `mkl`, you should additionally set `MKL_NUM_THREADS` to 1 and `MKL_DYNAMIC` to `FALSE`.

After startup, the first messages contain information about your computer’s environment: Code version, compiler information, host names, environment variables which turned out to be useful and which should be set on your system (e.g. `OMP_NUM_THREADS`), etc. The complete input files `control.in` and `geometry.in` are also repeated verbatim. Any FHI-aims run should thus be completely reproducible based on the standard output stream alone.

Should you encounter further issues, consider also the troubleshooting information documented in [Appendix A](#).

All this said, after successfully running the test run, you should now be ready to go with FHI-aims. The remainder of this document is about the details – available options, how to run aims most efficiently, etc. Happy computing!

## 1.4 Compilation options beyond the standard Makefile

The build provided by the standard Makefile in FHI-aims is designed for minimal complexity to obtain the full functionality that most users should have. Separate “basic linear algebra subroutines” (BLACS), Lapack, parallel builds (if a parallel machine is available, nowadays almost always), and scalapack support are so performance-critical that every user should spend the time to investigate them in detail before doing serious production work with FHI-aims. These dependencies of FHI-aims on external libraries are therefore kept in the main Makefile.

In addition, FHI-aims provides further functionality that can be achieved by linking to other external libraries. However, this functionality will not be needed by all users and/or could seriously complicate the build process for everyone. Such functionality is therefore available through separate, amended versions of the Makefile. We encourage everyone to try these builds (they are not so difficult after all), but they should not become stumbling blocks.

In particular, several optional Makefile with additional functionality exist (and could even be combined):

- `Makefile.cuba` : Allows to compile in the separate “CUBA” Monte Carlo integration library, which enables the Langreth-Lundqvist van der Waals functional based on `noloco` as a post-processing step. See Section 3.18 for more details.
- `Makefile.meta` : *still experimental!* Allows to interface FHI-aims to the PLUMED library for free-energy calculations for molecular dynamics. (see <http://merlino.mi.infn.it/~plumed/PLUMED/Home.html> , the PLUMED project homepage). Currently, a copy of the PLUMED library is kept in the *external* directory of the FHI-aims source code, and must be compiled separately using a C compiler. Note that the token “-DFHIAIMS” must be included in the `CCFLAGS`. For compiling on IBM power machines, the flags `-mpowerpc64 -maix64` should be included. In the future, this will be corrected by housing the respective FHI-aims plugin directly in the PLUMED library. We apologize that the linking process is not yet further documented, but if this functionality is of interest to you, please contact us.  
By selecting the correct target in `Makefile.meta`, the code can be compiled with `lapack` or `scalapack` libraries, or with shared memory support (see next item).
- `Makefile.wrappi` : Allows to interface with Michele Ceriotti’s `wrap-pi` path integral molecular dynamics wrapper, experimental at this point. See the [use\\_pimd\\_wrapper](#) keyword for a few more details.
- `Makefile.shm` : Another example of a Makefile that cross-links C and Fortran based functionality, although the actual functionality that `Makefile.shm` provides – access to shared-memory arrays in the Hartree potential – is no longer needed at this point.

## **Chapter 2**

### **Input Files: Basic Handling**

```
# Geometry for water -- needs to be relaxed as the water molecule
# described here has a 90degree bond angle and a
# 1 Angstrom bond distance ...
atom 0.00000000 0.00000000 0.00000000 O
atom 0.70700000 -0.70700000 0.00000000 H
atom -0.70700000 -0.70700000 0.00000000 H
```

Figure 2.1: Example input file `geometry.in`, provided with the simple testrun (relaxation of  $\text{H}_2\text{O}$ ) described in Sec. 1.3.

## 2.1 The mandatory input files: `control.in` and `geometry.in`

As discussed in Sec. 1.3, FHI-aims requires exactly two input files—`control.in` and `geometry.in`—located in the same directory from which the FHI-aims binary is invoked. To start FHI-aims, no further input should be needed.<sup>1</sup>

Figures 2.1 and 2.2 show as examples the `geometry.in` and `control.in` files used for the simple test case (relaxation of a water molecule) described in Sec. 1.3. The philosophy of their separation is simple:

- `geometry.in` contains only information directly related to the atomic structure for a given calculation. This obviously includes atomic positions, with a description of the particulars of each element (or *species*) expected in `control.in`. In addition, lattice vectors may be defined if a periodic calculation is required. Any other information is only given here if it is *directly* tied to the atom in question, such as an initial charge, initial spin moment, relaxation constraint etc. The order of lines is irrelevant, except that information specific to a given atom must follow *after* the line specifying that atom, and *before* any following atom is specified.
- `control.in` contains all other runtime-specific information. Typically, this file consists of a *general* part, where, again, the particular order of lines is unimportant. In addition, this file contains *species* subtags that are references by `geometry.in`. Within the description of a given species, the order of lines is again unimportant, but *all* information concerning the same species must follow the initial *species* tag in one block.

In both files, the choice of units is Å for length parameters, and eV for energies; derived quantities are handled accordingly. Lines beginning with a `#` symbol are treated as comments, and empty lines are ignored. Finally, each non-comment line has the following, free-format structure:

```
keyword value <value> <value>
```

<sup>1</sup>A few specific keywords (e.g., a wave-function based restart of an existing calculation) may require additional input that simply can not be included in user-edited file. Such input files will be described with the appropriate tasks.

```
#####
#
# Volker Blum, FHI 2009 : Test run input file control.in for simple H2O
#
#####
#
# Physical model
#
xc          pbe
spin        none
relativistic none
charge      0.
#
# SCF convergence
#
occupation_type gaussian 0.01
mixer         pulay
  n_max_pulay    10
  charge_mix_param 0.5
sc_accuracy_rho 1E-5
sc_accuracy_eev 1E-3
sc_accuracy_etot 1E-6
sc_accuracy_forces 1E-4
sc_iter_limit 100

[...]

#####
#
# FHI-aims code project
# VB, Fritz-Haber Institut, 2009
#
# Suggested "light" defaults for H atom (to be pasted into control.in file)
# Be sure to double-check any results obtained with these settings for post-processing
# e.g., with the "tight" defaults and larger basis sets.
#
#####
species      H
#   global species definitions
nucleus      1
mass         1.00794

[...]
```

Figure 2.2: Excerpts from the example input file `control.in`, provided with the simple testrun (relaxation of  $\text{H}_2\text{O}$ ) described in Sec. 1.3. A section of *general* (system-wide) runtime settings is separate from individual sections that describe settings specific to certain *species* (chemical elements).

Generally, all keywords and values are case sensitive: Do not expect FHI-aims to understand an “XC” keyword if the specified syntax is “xc”.

It is the objective of the *next* chapter, Chapter 3, to list all legitimate keywords in FHI-aims, and to describe their function.

## 2.2 Defaults for chemical elements: `species_defaults`

FHI-aims requires exactly two input files, located in the same directory where a calculation is started: `control.in` and `geometry.in`. Both files can in principle be specified from scratch for every new calculation, using the keywords listed in Chapter 3. However, choosing the central computational settings consistently for series of calculations greatly enhances the accuracy of any resulting energy differences (error cancellation).

In FHI-aims, the key parameters regarding computational accuracy are actually subkeywords of the `species` keyword of `control.in`, controlling the basis set, all integration grids, and the accuracy of the Hartree potential. These settings should of course not be retyped from scratch for every single calculation; on the other hand, they should remain obvious to the user, since these are the central handles to determine the accuracy and efficiency of a given calculation.

FHI-aims therefore provides preconstructed default definitions for the important subkeywords associated with different `species` (chemical elements) from  $Z=1-102$  (H-Md). These can be found in the `species_defaults` subdirectory of the distribution, and are built for inclusion into a `control.in` file by simple copy-paste.

In total, FHI-aims offers three different levels of `species_defaults` for each element:

- *light* Out-of-the-box settings for fast prerelaxations, structure searches, etc. In our own work, no obvious geometry / convergence errors resulted from these settings, and we now recommend them for many household tasks. For “final” results (meV-level converged energy differences between large molecular structures etc), any results from the *light* level should be verified with more accurate post-processing calculations, e.g. *tight*.
- *tight* : Regarding the integration grids, Hartree potential, and basis cutoff potentials, the settings specified here are rather safe, guaranteeing meV-level accurate energy differences also for large structures. For convergence purposes, the specification of the basis set itself (*tier 1*, *tier 2*, etc.) may still be decreased / increased as needed. In the *tight* settings, the basis set level is set to *tier 2* for the light elements 1-10, a modified *tier 1* for the slightly heavier Al, Si, P, S, Cl (the first *spdfgd* radial functions are enabled by default), and *tier 1* for all other elements. This reflects the fact that, for heavy elements, *tier 1* is sufficient for tightly converged ground state properties in DFT-LDA/GGA, but for the light elements (H-Ne), *tier 2* is, e.g., required for meV-level converged energy differences.
- *really\_tight* : *Strongly overconverged* settings for convergence verification beyond *tight*. We emphasize that the *really\_tight* settings should only ever be needed for individual, specific tests. They should not be needed for any standard production tasks unless you have seriously too much CPU time to spend. The basis sets and cutoff parameters are not increased compared to *tight* (this should be done by hand). However, the `basis_dep_cutoff` keyword is set to zero, a prerequisite to approach the converged basis limit. Regarding the Hartree potential, `l_hartree`

is set to 8, and the maximum number of angular grid points per radial integration shell is increased to 590.

In addition, a fourth level *light\_194* is provided for elements 1-10 (H-Ne), for very fast calculations (e.g., very long, rough molecular dynamics runs). Compared to the *light* settings, the only change is that the maximum number of angular integration points per radial grid shell is reduced from 302 to 194. For many tasks, for example the relaxation of the H<sub>2</sub>O molecule in the test cases provided for FHI-aims, this choice is still perfectly sufficient, but for consistency with the remaining *light* settings, these even lighter defaults are now kept separately.

For calculations that involve the excited state spectrum directly (this includes *GW*, *MP2*, or *RPA*, among others), we note that the numerical settings from *tight* should still perform very well *if* a counterpoise correction is performed, but the basis set size and/or cutoff radii *must* be converged and carefully verified beyond the settings specified in *tight*.

## 2.3 A very quick guide to ensuring numerical convergence with FHI-aims

FHI-aims is programmed and set up to allow efficient all-electron calculations for any type of system. During the writing of FHI-aims, a key goal was to always ensure that such efficiency does not come at the price of some irretrievable accuracy loss. Results obtained by FHI-aims should be *the* answer to the physical question that you asked (provided that the functionality is there in FHI-aims) - not some arbitrary approximation.

The *species\_default* levels provided by FHI-aims, *light*, *tight*, and (if ever needed!) *really\_tight*, should provide such reliable accuracy as they come. However, all important accuracy choices are here deliberately kept out in the open and available: They can—and sometimes should!—be explicitly tested by the user to check the convergence of a given calculation.

Such a convergence test may sometimes be geared at simply ensuring numerical convergence explicitly, but equally, it is possible that some default settings are too tight for a specific purpose, and can be relaxed in a controlled way to ensure faster calculations for some large problem.

In the following, we explain the most important species default settings explicitly, and comment on how to choose them. We use the *light* defaults for Oxygen as an example.

### 2.3.1 Basis set

The key physical choice to ensure converged results in FHI-aims is the list of radial functions (and their angular momenta) that are used in FHI-aims. Beyond the *minimal* basis of free-atom like radial functions, we always recommend to add at least a single set of further radial functions that are optimized to describe a chemical bond efficiently. These basis functions can be found as a list (line by line) at the end of each species defaults file. For Oxygen / *light*, the list reads like this:

```
# "First tier" - improvements: -699.05 meV to -159.38 meV
  hydro 2 p 1.8
  hydro 3 d 7.6
  hydro 3 s 6.4
# "Second tier" - improvements: -49.91 meV to -5.39 meV
#   hydro 4 f 11.6
#   hydro 3 p 6.2
#   hydro 3 d 5.6
#   hydro 5 g 17.6
#   hydro 1 s 0.75
# "Third tier" - improvements: -2.83 meV to -0.50 meV
#   ionic 2 p auto
#   hydro 4 f 10.8
#   hydro 4 d 4.7
#   hydro 2 s 6.8
```

[...]

Obviously, only a single set of radial functions (one for each angular momentum  $s$ ,  $p$ ,  $d$ ) is active (not commented!) beyond the minimal basis. Since the minimal basis already contains one additional valence  $s$  and  $p$  function, this choice is often called “double numeric plus polarization” basis set in the literature (where  $d$  is a so-called polarization function as it does not appear as a valence angular momentum of the free atom). We call this level “tier 1”.

In order to increase the accuracy of the basis, further radial functions may be added, simply by uncommenting more lines *in order*! We recommend to normally proceed in order of full “tiers”, not function by function, but adding specific individual functions on their own can sometimes capture the essence of a problem at lower cost. For example, tier 2 may be added by uncommenting:

```
# "First tier" - improvements: -699.05 meV to -159.38 meV
  hydro 2 p 1.8
  hydro 3 d 7.6
  hydro 3 s 6.4
# "Second tier" - improvements: -49.91 meV to -5.39 meV
  hydro 4 f 11.6
  hydro 3 p 6.2
  hydro 3 d 5.6
  hydro 5 g 17.6
  hydro 1 s 0.75
# "Third tier" - improvements: -2.83 meV to -0.50 meV
#   ionic 2 p auto
#   hydro 4 f 10.8
#   hydro 4 d 4.7
#   hydro 2 s 6.8
[...]
```

tier 2 is the default choice of our *tight* settings for O.

Beyond the choice of the radial functions itself, a critical parameter is the choice of the *confinement* radius that all basis functions experience. Ensuring that each radial function goes to zero in a controlled way beyond a certain, given value is critical for numerical efficiency, but on the other hand, you do not want to reduce this confinement radius too much in order to preserve the *accuracy* of your basis set.

By default, the confinement radius of each potential is specified by the following line:

```
cut_pot          3.5  1.5  1.0
```

This means (see also the CPC publication on FHI-aims) that each radial function is constructed with a confinement potential that begins 3.5 Å away from the nucleus, and smoothly pushes the radial function to zero over a width of 1.5 Å. The full extent of each radial function is thus 5 Å.

Of course, this setting is chosen to give good total energy accuracy at the *light* level, but the convergence of the confinement potential must still be tested, especially in situations where a strong confinement may be unphysical. Such questions include:

- Accurate free atom calculations for reference purposes: choose 8 Å or higher for the onset of the confinement, or something similarly high—for a single free atom, the CPU time will not matter, and you will get all the tails of your radial functions right without much thinking.
- Surfaces— e.g., low electron densities above the surface for STM simulations must not be abbreviated by the onset of the confinement potential—even if the total energy is not affected by this confinement any more.
- Neutral alkali atoms, or any negatively charged ions. Those are tricky—the outermost electron shell may decay very slowly to zero with distance, and explicit convergence tests are required.

As the corresponding *tight* setting, we use:

```
cut_pot          4.0  2.0  1.0
```

Although the modification does not seem large, CPU times for periodic systems are significantly affected by this change of the full extent of each radial function from 5 Å to 6 Å. For example, in a densely packed solid, the *density* of basis functions per volume increases as  $R^3$  with the full extent of each radial function, and thus the time to set up the Hamiltonian matrix should increase as  $R^6$ . Very often, the effect on the total energy is completely negligible, but again, explicit convergence tests are always possible to make sure.

Finally, there is the line

```
basis_dep_cutoff 1e-4
```

If this criterion is set above zero ( $10^{-4}$  in our *light* settings), all radial functions are individually checked, and their tails are cut off at a point where they are already near zero.

You should note that the `basis_dep_cutoff` criterion usually does not matter at all, but for very large systematic basis set convergence studies (going to tier 3, tier 4, etc, and/or testing the cutoff potential explicitly), this value should be set to zero—as is done in the *really\_tight* settings, for example.

### 2.3.2 Hartree potential

The Hartree potential in FHI-aims is determined by a multipole decomposition of the electron density. The critical parameter here is the order (highest angular momentum) used in the expansion (all higher components are neglected). This value is chosen by:

```
l_hartree      4
```

Energy differences with this choice are usually sub-meV converged also for large systems, but total energy differences, vibrational frequencies at the  $\text{cm}^{-1}$  level etc may require more. Our *tight* settings,

```
l_hartree      6
```

provide sub-meV/atom converged *total* energies in all our tests, but you may simply wish to test for yourself ...

### 2.3.3 Integration grid

FHI-aims integrates its Hamiltonian matrix elements numerically, on a grid. However, this is an all-electron code: Performing integrations on an *even-spaced* grid (as is done in many pseudopotential codes) would provide terrible integration accuracy near the nucleus (sharply peaked, deep Coulomb potential and strongly oscillating basis functions).

Instead, we use what is a standard choice also in other codes: Each atom gets a series of radial spheres (*shells*) around it, and we distribute a certain number of actual grid points on each shell. Obviously, increasing the number of grid points (“angular” points) on each shell will improve the integration accuracy, but at the price of a linear increase in computational cost.

The fact that the integration spheres will overlap does not matter—we remedy this fact automatically by choosing appropriate integration weights (partitioning of unity, see CPC paper).

The number and basic location of radial shells is chosen by

```
radial_base      36 5.0
radial_multiplier 1
```

which means that we here choose 36 grid shells, and the outermost shell is located at 5 Å (this happens to be the outermost radius of each basis function, as dictated by the confinement potential).

The `radial_multiplier` tag allows to increase the radial grid density systematically by adding shells inbetween those specified in the `radial_base` line. For example, we choose

```
radial_multiplier 2
```

in our *tight* species defaults (for all practical purposes, this is converged), which means that we add one shell between the zero and the (former) first shell, one between the first and second, etc., and finally one between the (former) outermost shell and infinity ... two times 36 plus one shells total.

The distribution of actual grid points *on* these shells is done using so-called Lebedev grids, which are designed to integrate all angular momenta up to a certain order  $l$  exactly. They come with fixed numbers of grid points (50, 110, 194, etc). As a rule, fewer grid points will be needed in the inner grid shells, and more will be needed at the (more extended) faraway grid shells. We specify the increase the number of grid points per radial shell in steps, by writing:

```

angular_grids specified
  division  0.2659  50
  division  0.4451 110
  division  0.6052 194
#   division  0.7543 302
#   division  0.8014 434
#   division  0.8507 590
#   division  0.8762 770
#   division  0.9023 974
#   division  1.2339 1202
#   outer_grid 974
outer_grid 194

```

This example pertains to the *light\_194* settings (and very light elements), and means that only 50 points will be used on all grid shells inside a radius of 0.2659 Å, 110 grid points are used on all shells within 0.4451 Å, 194 grid points will be used on all shells inside 0.6052 Å—and that's it! No more *division* tags are uncommented, and all shells outside 0.6052 Å also get 194 grid shells, as given by the uncommented *outer\_grid* tag.

We note that the form of the *increase* of the number of points per radial shell near the nucleus, as well as the *maximum* number of angular grid points used outside a given radius are *critical* for the numerical accuracy in FHI-aims. When suspecting numerical noise anywhere in the calculations, the specification of the angular grid points should be checked first. This can be done by uncommenting further *division* tags with larger numbers of grid points, as well as a suitably increased *outer\_grid* value. In particular, the choice of only 194 grid points max. per radial shell (only for the lightest elements!) is a rather aggressive choice, but in our experience still enables very reasonable geometry relaxations, structure searches or molecular dynamics for most purposes. However, the first thing to check in order to provide better convergence would be to set *outer\_grid* to 302 (regular *light* settings). If this produces a noticeable change of the quantity you are calculating, be careful.

Of course, one can always introduce the denser grids provided in the *tight* settings, which (for reference) are

```

angular_grids specified
  division  0.1817  50
  division  0.3417 110
  division  0.4949 194
  division  0.6251 302

```

```
division 0.8014 434
# division 0.8507 590
# division 0.8762 770
# division 0.9023 974
# division 1.2339 1202
# outer_grid 974
outer_grid 434
```

These grids alone are roughly twice as expensive as the *light\_194* ones above, and should provide reasonable accuracy for pretty much any purpose. Nonetheless, of course one can still go and check explicitly, simply by increasing the number of grid points per shell by hand.

## 2.4 Stopping a run: Files `abort_scf` and `abort_opt`

Sometimes, you may wish to stop a running FHI-aims calculation prematurely, but in an organized way.

Of course, with any running instance of FHI-aims, there is always the option to stop a run by invoking the Unix 'kill' command on every single running 'aims' process, and this will normally end the run right where it is.

To obtain a slightly more civilized stop (to allow the code to finish in a defined location and stop after writing some more output), you may instead create one of two specific files:

1. `abort_scf`
2. `abort_opt`

The code simply checks for the existence of either of these files periodically. No input is needed. Thus, simply change to the directory in which the code is running, and type (at the command line)

```
touch abort_scf
```

or

```
touch abort_opt
```

After a while, the run will stop.

The existence of `abort_scf` will stop the code after the current s.c.f. *iteration* is finished, i.e., the solution of the Kohn-Sham equations will not be self-consistent even for the present geometry.

The existence of `abort_opt` will stop the code after the current s.c.f. *cycle* is converged during a geometry relaxation, i.e., the electronic structure will be converged for the present geometry, but the forces will not be zero.

In either case, the stop of FHI-aims will not happen immediately. Depending on the nature of the run, it may take quite some time until the 'abort' takes effect, since the code needs to reach the appropriate state first. If you are interested in an immediate stop, the Unix 'kill' command is still your best bet.

One can also envision numerous refinements or alternative scenarios where an 'abort' file could be useful. If you really need such a case, please create the appropriate check where you need it. If it does the trick for you, we will be happy to incorporate the change into the mainline version of FHI-aims.

## Chapter 3

# The Full Monty: All Keywords and Capabilities

The present chapter aims to give a comprehensive overview and summary of all input options (keywords) that are available in FHI-aims: a full listing of keywords according to their intended use. In each of the following sections, keywords related to a given class of tasks are grouped together, and then listed according to whether they belong into `geometry.in`, the general section of `control.in`, or the species subsection(s) of `control.in`.

FHI-aims is a computer code under active development. Aside from established, stable and well-tested features, you may also find features that someone is still working on. Such features are marked as “experimental”. If you are interested in using one or more of those features, contact us, and we will try to be of assistance as much as we can.

For the truly curious: All input and output options are managed by the subroutines `read_control.f90`, `read_geo.f90`, and `read_species_data.f90`. In cases of doubt, those subroutines are the ultimate place to determine a keyword’s exact invocation and function.

## 3.1 Usability (convenience)

This section is only intended for functionality that fits none of the other categories (which are all scientifically / technically motivated). These files and keywords affect the general user convenience / experience for FHI-aims.

As an exception, this section also lists any *files* which may be used to interact with a *running* instance of FHI-aims. Currently, only two such files exist, but in principle, more could be envisioned.

### Files that interacting with the running code:

---

#### Tag: `abort_scf` (file)

Usage: At the command line, use the Unix command `touch abort_scf` in the current working directory of a running instance of FHI-aims to trigger a controlled stop of the run later.

Purpose: If the file `abort_scf` is found in the current working directory of FHI-aims, the present run will be aborted after the next s.c.f. iteration is complete (but importantly without achieving self-consistency).

This functionality allows FHI-aims to stop in a controlled fashion, but not instantly. If you are interested in an instant stop, the Unix 'kill' command (or its equivalent in the queueing system of a production machine) is the best way to proceed. See Sec. 2.4 for some further remarks.

---

#### Tag: `abort_opt` (file)

Usage: At the command line, use the Unix command `touch abort_opt` in the current working directory of a running instance of FHI-aims to trigger a controlled stop of the run later.

Purpose: If the file `abort_opt` is found during a geometry relaxation in the current working directory of FHI-aims, the present run will be aborted after the next s.c.f. cycle is complete (i.e., after achieving self-consistency for the present geometry, but without fully optimizing the structure).

This functionality allows FHI-aims to stop in a controlled fashion, but not instantly. If you are interested in an instant stop, the Unix 'kill' command (or its equivalent in the queueing system of a production machine) is the best way to proceed. See Sec. 2.4 for some further remarks.

---

## Tags for general section of `control.in`:

---

**Tag:** `dry_run` (`control.in`)

Usage: `dry_run`

Purpose: If set in `control.in`, the FHI-aims run will only pass through all preparatory work to ensure the basic consistency of all input files, but will stop before any real work is done.

This keyword is useful to check the consistency of input files with the same exact binary that may later be used in a series of (perhaps queued) production runs. If there are trivial errors in the input files, no need to wait for the queue. The same effect can be achieved by building a 'parser' binary, but this version saves the recompilation. The price is that one must not forget to comment out the `dry_run` option in the actual, queued input files.

## 3.2 Physical model: Geometry, charge, spin, etc.

The present section summarizes all keywords in FHI-aims that are directly concerned with the *physical model* of the problem to be tackled. Importantly, this includes some specific subtags that you *cannot* ignore, because they define the physical question that you are trying to address – and no one else but you can do that. The present section thus includes such things as atomic positions or unit cells, but also the level of theory to be used (exchange-correlation, relativistic treatment), or a potential charge of the system.

### Tags for `geometry.in`:

---

#### Tag: `atom` (`geometry.in`)

Usage: `atom x y z species_name`

Purpose: Specifies the initial location and type of an atom.

$x$ ,  $y$ ,  $z$  are real numbers (in Å) which specify the atomic position.

`species_name` is a string descriptor which names the element on this atomic position; it must match with one of the species descriptions given in `control.in`.

---

#### Tag: `atom_frac` (`geometry.in`)

Usage: `atom_frac n1 n2 n3 species_name`

Purpose: Specifies the initial location and type of an atom in fractional coordinates.

$n_i$  is a real multiple of lattice vector  $i$ . `species_name` is a string descriptor which names the element on this atomic position; it must match with one of the species descriptions given in `control.in`.

Fractional coordinates are only meaningful in peridic calculations.

---

#### Tag: `lattice_vector` (`geometry.in`)

Usage: `lattice_vector x y z`

Purpose: Specifies one lattice vector for periodic boundary conditions.

$x$ ,  $y$ ,  $z$  are real numbers (in Å) which specify the direction and length of a unit cell vector.

If up to three lattice vectors are specified, FHI-aims automatically assumes periodic boundary conditions in those directions. *Note* that the order of lattice vectors matters, as the order of  $k$  space divisions (given in `control.in`) depends on it!

---

## Tags for general section of control.in:

---

### Tag: charge (control.in)

Usage: `charge q`

Purpose: If set, specifies an overall charge in the system.

`q` is a real number that specifies a positive or negative total charge in the system.

For most normal systems, this definition is unambiguous (sum of all nuclear charges in `geometry.in` minus number of electrons in the system). Note specifically that the same definition continues to hold also in systems with external embedding charges (specified by keyword `multipole` in `geometry.in`). The charges of the external embedding charges are in addition to the `charge` keyword in `control.in`, and *not* included.

---

### Tag: fixed\_spin\_moment (control.in)

Usage: `fixed_spin_moment value`

Purpose: If set, allows to enforce a fixed overall spin moment throughout the calculation.

`value` : real-valued number, specifies the difference of electrons between spin channels,  $2S = N_{\text{up}} - N_{\text{down}}$ .

Meaningful only in the spin-polarized case (`spin collinear` in `control.in`).

This keyword replaces the earlier keyword `multiplicity`. Note that the value that must be given for `fixed_spin_moment` is  $2S$ , which corresponds to a `multiplicity` ( $2S + 1$ ) —i.e., the values are not the same. Keyword `fixed_spin_moment` works for periodic and cluster systems alike, and uses two different chemical potentials (Fermi levels) for the spin channels.

---

### Tag: species (control.in)

Usage: `species species_name`

Purpose: Defines the name of a species (element) for possible use with atoms in `geometry.in`

`species_name` is a string descriptor (e.g. C, N, O, Cu, Zn, Zn\_tight, ...).

Every `species_name` used in an atom descriptor in `geometry.in` must correspond to a `species` given in `control.in`. Following the `species` tag, all sub-tags describing that species must follow in one block. (No particular order is enforced within that block). For example, the choice of the basis set, the atom-centered integration grid, or the multipole decomposition of the atom-centered Hartree potential are all specified per `species`.

---

### Tag: spin (control.in)

Usage: `spin` type

Purpose: Specifies whether or not spin-polarization is used.

`type` is a string, either `none` or `collinear`, depending on whether an unpolarized (spin-restricted) or spin-polarized (unrestricted) calculation is performed.

In the `collinear` case, defining the moments used to create the initial spin density is required (see the beginning of Sec. 3.10 for an explanation). This means that an overall `default_initial_moment` (in `control.in`), or at least one individual `initial_moment` tag in `geometry.in`, or both, must be set. Else, the code will stop with a warning. (It is not necessary to specify `initial_moment` for every atom in `geometry.in`. A single one will do.) Choosing the right initial spin density can be performance-critical, and critical for the resulting physics. FHI-aims should not make this choice for you.

---

## Subtags for *species* tag in `control.in`:

---

### `species` sub-tag: `mass` (`control.in`)

Usage: `mass M`

Purpose: Atomic mass

M is a real number that specifies the atomic mass in atomic mass units (amu).

This tag is used only for molecular dynamics. The preconstructed `species_defaults` files supplied with FHI-aims contain the mass *average* over the different isotopes of each natural element.

---

### `species` sub-tag: `nucleus` (`control.in`)

Usage: `nucleus Z`

Purpose: Via the nuclear charge defines the chemical element associated with the present species.

Z is an integer number (the nuclear charge).

### 3.3 Electronic structure: Exchange, correlation (incl. DFT+U), and excited states

A key choice *required* in every electronic structure calculation is the treatment of the required electronic structure: Exchange, correlation, and potentially quasiparticle energies, e.g., after a *GW* correction.

We here summarize the general options available regarding the choice of the electronic structure method. In addition, an important question is *which* electrons in the structure are treated at which level. For most practical purposes, FHI-aims treats all electrons in an equivalent way, but for some special cases, frozen-core treatments may be useful: at present, one may compute the correlation energy of only the *valence* but not the *core* electrons in second-order Møller-Plessett (MP2) perturbation theory.

For any method requiring the two-electron Coulomb operator explicitly (these include hybrid functionals, Hartree-Fock, MP2 or RPA perturbation theory, *GW* corrections, etc.) we note that an auxiliary basis is required to expand the Coulomb matrix (four basis functions  $\equiv O(N^4)$  matrix elements) into a two-center Coulomb matrix, leading instead to  $O(N^3)$  additional overlap matrix elements. The choice of this auxiliary basis (“product basis”) is described in more detail in Sec. 3.19 and Ref. [95].

#### A note on “post-s.c.f” RPA-based methods

The algorithms for post-DFT methodologies as implemented in FHI-aims are detailed in Ref. [95]. Here we only briefly recapitulate the key ingredients behind the increasingly popular “RPA and beyond” methods as implemented in FHI-aims. The standard RPA total energy is computed as follows:

$$E_{\text{tot}}^{\text{RPA}} = E_{\text{tot}}^{\text{DFT}} - E_{\text{xc}}^{\text{DFT}} + E_{\text{x}}^{\text{EX}} + E_{\text{c}}^{\text{RPA}}. \quad (3.1)$$

$E_{\text{tot}}^{\text{DFT}}$  is a pre-computed self-consistent DFT total energy obtained from LDA, GGA, or hybrid functional calculations.  $E_{\text{xc}}^{\text{DFT}}$  is the corresponding exchange-correlation contribution.  $E_{\text{x}}^{\text{EX}}$  and  $E_{\text{c}}^{\text{RPA}}$  are the exact-exchange energy, and the RPA non-local correlation evaluated using the pre-determined Kohn-Sham or generalized Kohn-Sham eigenorbitals and eigenenergies.

Recently, several correction schemes to RPA have been proposed. FHI-aims currently provides the (renormalized) single excitation (SE) correction [97] and the second-order screened exchange (SOSEX) correction [38]. The renormalized SE (rSE) and SOSEX corrections can be combined. The combined scheme is called “renormalized 2nd-order perturbation theory” (rPT2) [96],

$$E_{\text{tot}}^{\text{rPT2}} = E_{\text{tot}}^{\text{RPA}} + E_{\text{c}}^{\text{SOSEX}} + E_{\text{c}}^{\text{rSE}}. \quad (3.2)$$

The “RPA+SE”, “RPA+rSE”, and “RPA+SOSEX” total energies can be computed similarly by combining the corresponding terms.

#### A note on “DFT plus U”

In the DFT method with local or semi-local approximations of the XC-functional, (LDA, GGA, etc.) strongly correlated systems like transition metal oxides are poorly described.

The “DFT plus U” method offers an ad hoc correction for strongly correlated systems at negligible computation cost [4].

*The present implementation of “DFT plus U” in FHI-aims should be considered experimental, and is not complete in some respects. Please keep this in mind when using the method. That said, it should give physically sensible results. Simply take some care when using it, and please give us feedback if the method works for you (obviously, also if it does not for some reason). Thanks go to Dr. Norbert Nemeč for preparing the present implementation.*

- DFT+U total energies can be obtained in combination with any functional (typically LDA or GGA), by simply adding appropriate `plus_u` tags to the corresponding species.
- Total energy gradients (“forces”) are not provided.
- The implementation does not yet offer self-consistent determination of the U parameter, so this needs to be supplied by hand.
- Finally, the orbitals on which we project are the somewhat extended free-atom like orbitals, defined with the usual cutoff potential of the remaining calculation. While somewhat arbitrary, it would be useful to be able to project onto more localized orbitals, but this option is not implemented yet.

## Tags for general section of `control.in`:

---

**Tag:** `frozen_core` (`control.in`)

Usage: `frozen_core` `first_orbital`

Purpose: Allows to compute the MP2 correlation energy without the contribution arising from low-lying occupied orbitals.

`first_orbital` is the integer number of the first molecular orbital that is *included* in the computation of the MP2 correlation energy.

**This keyword applies only to the calculation of the MP2 correlation energy (if requested). It does not imply a frozen-core treatment anywhere else.**

In a nutshell, this is a simple way to exclude the large contribution from certain core electrons to the MP2 correlation energy. This contribution is mostly systematic, and therefore tends to cancel in energy differences. However, it is also the hardest to compute unless specialized basis sets are invoked that “know” about core correlation; it may thus be the source of a large systematic error that also cancels if excluded from the beginning. For consistency between different calculations, the number of excluded “core” orbitals must be readjusted between calculations with different numbers of atoms.

---

**Tag:** `hybrid_xc_coeff` (`control.in`)

Usage: `hybrid_xc_coeff` value

Purpose: If set, will modify the (Hartree-Fock) exact exchange mixing parameter in a given hybrid XC functional. *No effect* if specified with a simple LDA / GGA type functional.

value is a real number (usually between zero and one) that specifies the degree of exact exchange admixture.

If (and only if) a hybrid functional is specified using the `xc` keyword, `hybrid_xc_coeff` allows to change the Hartree-Fock mixing parameter to a different, given value. For example, the mixing parameter in `pbe0` could be specified away from its literature value,  $\alpha=0.25$ . No effect for `xc` functionals that do not have any Hartree-Fock exchange admixed in the first place.

Obviously, this option is only useful for test purposes and does change the definition of any functional away from its literature value. Handle with care.

**Tag:** `hse_unit` (`control.in`)

Usage: `hse_unit` character

Purpose: Required clarification of units for the `hse06` `xc` functional.

value is a character, either 'a' or 'A' (for  $\text{\AA}^{-1}$ ) or 'b' or 'B' (for  $[\text{bohr radius}]^{-1}$ ).

The `hse06` functional comes with a screening parameter  $\omega$  which must be specified explicitly (see the `xc` keyword for a detailed explanation). Unfortunately, different codes and authors appear to have adopted different conventions for  $\omega$  – either  $\text{\AA}^{-1}$  or  $[\text{bohr radius}]^{-1}$ . To avoid any possible confusion when using HSE06 in FHI-aims, we therefore only run `hse06` if the unit has been explicitly specified, using the above keyword. We apologize for the inconvenience, but the risk of an innocent misunderstanding is rather high in the present case.

**Tag:** `plus_u_petukhov_mixing` (`control.in`)

Usage: `plus_u_petukhov_mixing` mixing\_factor

Purpose: *Experimental—only for DFT+U*. Allows to fix the mixing factor between AMF and FLL contribution of the double counting correction [90].

mixing\_factor is a floating point value, specifying the mixing ratio between 0.0 and 1.0. A value of 0.0 selects the Around Mean Field (AMF) contribution. A value of 1.0 selects the Fully Localized Limit (FLL). If unspecified, the value is determined self-consistently according to Ref. [90].

There are two common schemes for dealing with the double counting problem in DFT+U: The AMF method assumes that the effect of the DFT+U term on the actual occupations remains small, so that the occupations can be assumed to be equal within each shell for the purpose of the double counting correction. The FLL method, on the other hand, assumes a maximal effect of the DFT+U term on the occupation numbers, handling

double counting correctly in the case that all orbitals within the shell are either fully occupied or empty. The self-consistent mixing of both limits improves the handling of the intermediate range (see Ref. [90]).

---

**Tag:** `qpe_calc` (`control.in`)

Usage: `qpe_calc` `selfenergy-type`

Purpose: If set, specifies which self-energy should be used for a quasiparticle correction of single-particle eigenvalues.

`selfenergy-type` is a keyword (string) which specifies the selfenergy approximation used.

*Note* that quasiparticle corrections ( $GW$ , MP2) are currently possible only for cluster geometries (no periodic boundary conditions).

After the normal self-consistency cycle for a given exchange-correlation functional (set using the `xc` keyword) is complete, `qpe_calc` can be used to specify a perturbative quasiparticle correction to be applied as a post-processing step. Valid self-energy options `selfenergy-type` are:

- `gw` : Perturbative  $G_0W_0$ -type self-energy, where both the Green's function  $G_0$  and the screened Coulomb interaction  $W_0$  are computed only once, based on the self-consistent DFT or Hartree-Fock ground state eigenvalues and eigenfunctions.
- `ev_scgw` Perturbative  $G_0W_0$ -type self-energy, where self-energy is evaluated with partial self-consistency in the eigenvalues. Molecular orbitals are kept unchanged from the preliminary calculation. For true self-consistent  $GW$ , see the `sc_self_energy` further below.
- `mp2` : Perturbative MP2-type self-energy, based on the self-consistent DFT or Hartree-Fock ground state eigenvalues and eigenfunctions.

---

**Tag:** `sc_self_energy` (`control.in`)

Usage: `sc_self_energy` `self-consistent-scheme`

Purpose: If set, specifies the scheme adopted for the self-consistent calculation of the many-body self-energy.

`selfenergy-type` is a keyword (string) which specified the self-consistent approach used in the calculation.

*Note* that self-consistent  $GW$  calculation ( $sc-GW$ ,  $sc-GW_0$ ) are currently possible only for cluster geometries (no periodic boundary conditions).

After the normal self-consistency cycle for a given exchange-correlation functional (set using the `xc` keyword) is complete, `sc_self_energy` can be used to specify a self-consistent scheme for the calculation of the  $GW$  self-energy. The output consists of the total energy calculated from the Galitskii-Migdal formula, an output file (`spectrum_sc.dat`

for spin unpolarized, `spectrum_sc_up.dat` and `spectrum_sc_do.dat` for spin up and down respectively in the case of spin polarized calculation) containing the spectral function calculated from the self-consistent Green's function. At the end of the calculation, the output include the dipole moment evaluated from the self-consistent density.

Currently implemented self-consistent methods are:

- `scgw` : Calculate the Green's function by solving until full self-consistency the Dyson's equation by using a self-energy in the  $GW$  approximation.
- `scgw0` : Solve self-consistently the Dyson's equation with the self-energy in the  $GW_0$  approximation. Differently from fully self-consistent  $GW$ , in this case the screened Coulomb interaction is kept fixed at the RPA level.

**Tag:** `scgw_mix_param` (`control.in`)

Usage: `scgw_mix_param`  $\alpha$

Purpose: Define the linear mixing coefficient  $\alpha$ , for the mixing of the Green function at each iteration of the self-consistent  $GW$  calculation. This keyword only produces an effect if `sc_self_energy` is set.

**Tag:** `scgw_it_limit` (`control.in`)

Usage: `scgw_it_limit`  $N$

Purpose: Set the maximum number  $N$  of iteration of the Dyson equation in a self-consistent  $GW$  calculation. The default value is set to  $N = 30$ . This keyword only produces an effect if `sc_self_energy` is set.

**Tag:** `scgw_print_all_spectrum` (`control.in`)

Usage: `scgw_print_all_spectrum`

Purpose: Enables the print out of the spectral function each iteration of the self-consistent  $GW$  calculation. The spectrum is printed to the file `sp_ImG<N>.dat`, where `<N>` is number of iteration of the Dyson equation. This keyword only produces an effect if `sc_self_energy` is set.

**Tag:** `scs_mp2_parameters` (`control.in`)

Usage: `scs_mp2_parameters` pT pS

Purpose: For MP2 correlation energies, allows to perform spin-component scaled MP2.

pT is the scaling parameter for the spin-up-spin-up (triplet) contribution.

pS is the scaling parameter for the spin-up-spin-down (singlet) contribution.

The MP2 correlation energy (`total_energy_method mp2` or `xc mp2`) can be separated into a sum of triplet (spin-up-spin-up) and singlet (spin-up-spin-down) two-electron terms:

$$E_{\text{corr,MP2}} = E_T + E_S. \quad (3.3)$$

Grimme [37] pointed out that empirical scaling factors  $p_T$  and  $p_S$  can be introduced and fitted to improve the accuracy of MP2 results compared to quantum-chemical benchmark methods:

$$E_{\text{SCS,MP2}} = p_T E_T + p_S E_S. \quad (3.4)$$

For example,  $p_T=1/3$  and  $p_S=6/5$  are employed to obtain the reaction energies of Table I in Ref. [37].

**Tag:** `total_energy_method` (`control.in`)

Usage: `total_energy_method` type

Purpose: If set, specifies an exchange-correlation method *for post-processing only*, after the scf cycle is complete.

type is a keyword (string) which specifies the chosen post-processing exchange-correlation method.

After the regular scf cycle is complete for a given exchange-correlation method as given by the `xc` tag, the resulting Kohn-Sham orbitals and eigenvalues are used to recalculate *only* the exchange-correlation energy, and only once (i.e., perturbative post-processing). Valid post-processing options type are:

- `C6_coef` : Molecular  $C_6$  dispersion coefficients at the MP2 / RPA level will be calculated after the s.c.f. calculation. (This functionality is somewhat experimental, be sure to check for consistency.)
- `hf` or `HF`: Calculate Hartree-Fock exchange on the given orbitals.
- `11_vdwdf` : The nonlocal part of correlation energy is calculated using the van der Waals density functional proposed by M. Dion *et al.* [24] and the total correlation energy will be re-evaluated as proposed in their paper. For details about additional Tags needed for the calculation, please visit Sec. 3.18. Note that an alternative implementation by the Helsinki group is available as well, the present keyword is not your only option.
- `m06` or `M06`: Truhlar's modified hybrid meta-GGA of the "M06" suite of functionals (exact exchange contribution doubled). Currently supported only for non-hybrid functionals specified through the `xc` tag.
- `m061` or `M06L`: Truhlar's optimized local meta-GGA of the "M06" suite of functionals.
- `m06-2X` or `M06-X`: Truhlar's optimized hybrid meta-GGA of the "M06" suite of functionals. Currently supported only for non-hybrid functionals specified through the `xc` tag.

- `mp2` : The correlation energy is calculated in second-order Møller-Plesset perturbation theory (MP2), with Hartree-Fock added for the exchange part.
- `pbe_vdw` : Evaluates the van der Waals density functional proposed by M. Dion *et al.* [24] with the methodology of Sec. 3.18.2. (Uses PBE exchange.)
- `revpbe_vdw` : As `pbe_vdw` but uses `revpbe` instead of `pbe` for the exchange.
- `revtpss` : Meta-GGA revTPSS functional, thanks to E. Fabiano and F. Della Sala.
- `n1corr` : Only the non-local correlation term of the `pbe_vdw` or `revpbe_vdw` is calculated and added to the total energy.
- `rpa` : The RPA total energy as defined in Eq. (3.1) will be calculated. When this option is specified, the SE and rSE corrections to RPA are also evaluated. The total energies computed with the RPA, RPA+SE, and RPA+rSE schemes are listed in items "RPA total energy", "RPA+SE total energy", and "RPA+rSE (full) total energy" respectively in the output file.
- `rpa+2ox` : Just RPA plus second-order exchange (not screened). Likely only useful for testing / benchmarking, use `rpt2` for completeness.
- `rpa+sosex` : Just RPA plus second-order screened exchange. Likely only useful for testing / benchmarking, use `rpt2` for completeness.
- `rpt2` : The rPT2 total energy as defined in Eq. (3.2) will be calculated. When this option is specified, the "RPA+SOSEX" total energy without the rSE correction will also be printed out in the output file.
- `tpss` : Meta-GGA TPSS functional, thanks to E. Fabiano and F. Della Sala.
- `tpsslloc` : Meta-GGA TPSSloc functional, thanks to E. Fabiano and F. Della Sala. Reference: L.A. Constantin, E. Fabiano, F. Della Sala, Phys. Rev. B, 86, 035130 (2012).
- `xyg3` : "XYG3" doubly hybrid functional. Currently defined only for a self-consistent B3LYP reference, i.e., `xc b3lyp` is mandatory.

Note that some of the correlation methods available here are only supported for cluster geometries at this time. Note also that when `rpa` or `mp2` is used for binding energy calculations, a **counterpoise correction** should always be performed to get reliable results, since the basis set superposition error (BSSE) for these correlation methods is significant.

---

**Tag:** `use_2d_corr` (`control.in`)

Usage: `use_2d_corr` bool

Purpose: Specifies whether to use the efficient 2D distribution of the MO based three index arrays where possible. Otherwise, stick to the old 1D distribution in all cases.

Default: `.true.`

---

**Tag:** `xc` (`control.in`)

Usage: `xc xc-type [value]`

Purpose: Specifies the exchange-correlation approach used for self-consistent DFT / Hartree-Fock.

Default: `pw-lda`

`xc-type` is a keyword (string) which specifies the chosen exchange-correlation functional.

`value` is a real parameter needed only for some functionals (e.g., `hse06`).

FHI-aims provides a wide range of current exchange-correlation options, ranging from local-density and generalized-gradient approximations (LDAs and GGAs) via hybrid functionals and Hartree-Fock to two-electron treatments of the correlated many-body system, such as second-order Møller-Plesset (MP2) theory and the random-phase approximation (RPA). The following choices for the `xc-type` option are currently available:

- Local-density approximations:
  - `pw-lda` : Homogeneous electron gas based on Ceperley and Alder [16] as parametrized by Perdew and Wang 1992 [87]. *Recommended LDA*.
  - `pz-lda` : Homogeneous electron gas based on Ceperley and Alder [16], as parametrized by Perdew and Zunger 1981 [88].
  - `vwn` : LDA of Vosko, Wilk, and Nusair 1980 [111].
  - `vwn-gauss` : LDA of Vosko, Wilk, and Nusair 1980, *but based on the random phase approximation* [101]. Do not use this LDA unless for one specific reason: In the B3LYP implementation of the Gaussian code, this functional is allegedly used instead of the correct VWN functional. It is therefore now present in many reference results in the literature, and also available here for comparison.
- Generalized-gradient approximations:
  - `am05` : GGA functional designed to include surface effects in self-consistent density functional theory, according to Armiento and Mattsson [5]
  - `blyp` : The BLYP functional: Becke (1988) exchange [9] and Lee-Yang-Parr correlation [69].
  - `pbe` : GGA of Perdew, Burke and Ernzerhof 1997 [86].
  - `pbeint` : PBEint functional of Ref. [30]

- `pbesol` : Modified PBE GGA according to Ref. [89].
- `rpbe` : The RPBE modified PBE functional according to Ref. [42].
- `revpbe` : The revPBE modified PBE GGA suggested in Ref. [114].
- Hybrid functionals (including non-local exchange): *presently available for cluster-type geometries only, not for periodic systems!*
  - `b3lyp` : “B3LYP” hybrid functional as allegedly implemented in the Gaussian code (i.e., using the RPA version of the Vosk-Wilk-Nusair local-density approximation, see Refs. [111, 101] for details).
  - `hse03` : Hybrid functional as used in Heyd, Scuseria and Ernzerhof [48, 49]. In this functional, 25 % of the exchange energy is split into a short-ranged, screened Hartree-Fock part, and a PBE GGA-like functional for the long-range part of exchange. The remaining 75 % exchange and full correlation energy are treated as in PBE. As clarified in Refs. [62, 49], two different screening parameters were used in the short-range exchange part and long-range exchange part of the original HSE functional, respectively:  
Screened Hartree-Fock exchange:  $\omega_{\text{HF}} = 0.15/\sqrt{2}$   
Screened PBE-like exchange:  $\omega_{\text{PBE}} = 0.15 \times 2^{1/3}$   
Following the notation of Ref. [62], the ‘hse03’ functional in FHI-aims reproduces these original values exactly.
  - `hse06` : Hybrid functional according to Heyd, Scuseria and Ernzerhof [48], following the naming convention suggested in Ref. [62]. In this case, the additional option `value` is needed, representing the single real, positive screening parameter  $\omega$  as clarified in Ref. [62]. In this functional, 25 % of the exchange energy is split into a short-ranged, screened Hartree-Fock part, and a PBE GGA-like functional for the long-range part of exchange. The remaining 75 % exchange and full correlation energy are treated as in PBE. *In the literature, the unit for  $\omega$  is either  $\text{\AA}^{-1}$  or  $(\text{bohr radius})^{-1}$ , depending on the code, authors, and their favorite convention. To avoid any confusion, a separate keyword `hse_unit` must be specified in `control.in`, specifying either  $\text{\AA}^{-1}$  (‘A’) or  $\text{bohr}^{-1}$  (‘b’). The code will no longer run without this explicit clarification. A correct calling syntax example is therefore:*  

```
xc hse06 0.11
hse_unit bohr-11
```

or similar.  
A few comments on typical choices for  $\omega$  in the earlier literature:  
The original value of  $0.15 \text{ bohr}^{-1}$  by Heyd, Scuseria and Ernzerhoff 2003 [48] was never true - see their 2006 erratum. In FHI-aims, the ‘hse03’ functional implements their actual choice.  
Krukau, Vydrov, Izmaylov and Scuseria 2006 [62] clarify the distinction between ‘hse03’ and ‘hse06’ (in addition to the Erratum mentioned above). Their conclusion is that  $\omega=0.11 \text{ bohr}^{-1}$  is a reasonable choice.  
Vydrov, Heyd, Krukau and Scuseria in 2006 [84] appear to favor  $\omega=0.25$

<sup>1</sup>The `hse_unit` flag reads only the first character. Thus this is equivalent to `hse_unit b` (case insensitive).

bohr<sup>-1</sup>, but with a mixing parameter (keyword `hybrid_coeff`) of 0.5 for the short-range exchange. (The default for `hybrid_coeff` in FHI-aims is 0.25, i.e., only a quarter of HF-like exchange.)

You get the idea. As much as we would like to, we can not specify a single omega parameter for hse06 by default – the choice is up to you. Apologies for the inconvenience.

- `pbe0` : PBE0 hybrid functional [1], mixing 75 % GGA exchange with 25 % Hartree-Fock exchange.
- `pbesol` : Hybrid functional in analogy to PBE0 [1], except that the PBEsol [89] GGA functionals are used, mixing 75 % GGA exchange with 25 % Hartree-Fock exchange.

- Wave-function based methods (Hartree-Fock and beyond): *presently available for cluster-type geometries only, not for periodic systems!*

- `hf` : Hartree-Fock exchange only.
- `mp2` : Self-consistent Hartree-Fock, followed by a second-order Møller-Plesset perturbative addition of the correlation energy. Note that the `frozen_core` keyword can be used to specify if and which low-lying states should be excluded from the correlation energy. For spin-component scaled MP2 [37], see keyword `scs_mp2_parameters`.

*Note* that when `mp2` is used for binding energy calculations, a **counterpoise correction** should always be performed to get reliable results, since the basis set superposition error (BSSE) for these correlation methods is significant.

- `screx` : *experimental!* Self-consistent, screened Hartree-Fock exchange only. The Coulomb operator is screened as:

$$\frac{1}{r-r'} \rightarrow \frac{1}{\varepsilon(r,r')} \cdot \frac{1}{r-r'} \quad (3.5)$$

$\varepsilon(r,r')$  is the non-local *microscopic* dielectric function, obtained in the  $\omega \rightarrow 0$  frequency limit of the random-phase approximation (RPA). See Ref. [45] for details.

- `cohsex` : *experimental!* Self-consistent screened exchange plus Coulomb-hole (COH) correlation. See Ref. [45] for details.

- Method of non-local correlation using van der Waals density functional as presented in [24]. Two options are available for the exchange part:

- `pbe_vdw` : the functional with pbe exchange
- `revpbe_vdw` : the functional with revpbe exchange

*Note* that our version of the Coulomb operator (which is the basis for Hartree-Fock exchange also in hybrid functionals, as well as MP2 theory) is based on an auxiliary basis in what is known as *resolution of the identity* (Refs. [14, 2, 108, 28] and others). While our default settings should be safe, you may wish to consult Sec. 3.19 for particulars regarding this auxiliary basis.

Note also that some different *perturbative* exchange-correlation treatments for *post-processing* (after a self-consistent DFT or HF calculation is complete) may be invoked using the tag `total_energy_method`. Likewise, perturbative postprocessing for single-quasiparticle energies through a self-energy formalism (e.g., *GW*) is reached by specifying the `qpe_calc` tag and its options.

Right now, the correlated beyond-hybrid and beyond-meta methods are not implemented on top of the HSE03 or HSE06 functionals.

### Subtags for `species` tag in `control.in`:

---

#### `species` sub-tag: `plus_u` (`control.in`)

Usage: `plus_u n l U`

Purpose: *Experimental—only for DFT+U*. Adds a +U term to one specific shell of this species.

`n` the (integer) radial quantum number of the selected shell.

`l` is a character, specifying the angular momentum ( *s*, *p*, *d*, *f*, ...) of the selected shell.

`U` the value of the U parameter, specified in eV.

This implementation of DFT+U is based directly on the basis functions available within FHI-aims. This option selects one specific *atomic* shell of this species and adds the a rotationally invariant term with the specified fixed prefactor U to the Hamiltonian. The implementation follows the prescription in Ref. [43], based on the *dual* occupation numbers. The double counting term is handled through the mixed term proposed by Petukhov (see `plus_u_petukhov_mixing`).

## 3.4 Specifying the basis (functions, empty sites, k-points, ...)

Among the technical choices in FHI-aims, the choice of the basis set is by far the most important one, both regarding the efficiency and the desired accuracy of a calculation. The shape and details of the basis sets used are thus kept as obvious as possible to the user. At the same time, nobody should be required to type in an entire basis set plus additional specifications from scratch just to run a production calculation.

As described in Ref. [12], the basis functions of FHI-aims take the format

$$\phi(\mathbf{r}) = u(r)/r \cdot Y_{lm}(\theta, \phi) \quad (3.6)$$

in spherical coordinates  $(r, \theta, \phi)$  with respect to a given atomic center. Each radial function  $u(r)$  is numerically tabulated on a dense logarithmic radial grid, and evaluated as a cubic spline function in other parts of the code. Finally, most radial function types are subject to a cutoff potential of radial with  $w$ , ensuring that  $u(r)=0$  for  $r > r_{\text{cut}} = r_{\text{onset}} + w$ .

In periodic calculations, the full basis specification additionally includes the  $k$ -point grid for Bloch functions in the first Brillouin zone. Unlike in many other implementations, this is *not* a performance-critical setting in FHI-aims, and should be set to a well converged value if possible.

The recommended approach to basis sets in FHI-aims is twofold:

- *First*, obtain the basic description of each required element by copy-pasting one of the preconstructed `species_default` files into your `control.in` file. The preconstructed `species_default` files address all standard specifications associated with a single `species`, including the integration grids, the Hartree potential, and most importantly the basis set.
- *Second*, **edit the copy-pasted `species_defaults` file** to match your specific accuracy and efficiency requirements. For the basis set, this is done by adjusting the species-dependent keywords described below. Most importantly, *complete* basis sets are listed at the end of each `species_default` file. You can increase/decrease the basis set accuracy by successively uncommenting / commenting *tiers* of the basis set. Note that each higher *tier* must only be used if *all* lower *tiers* are active. For example, it does *not* make sense to use all tier 2 basis functions if the first tier is not used.

## Tags for general section of `geometry.in`:

---

**Tag:** `empty` (`geometry.in`)

Usage: `empty` x y z species\_name

Purpose: Specifies the initial location and type of a *site* where *only* the basis functions (but not the nucleus) of a given species are placed.

Restriction: Currently not functional with periodic boundary conditions. The use of this option should be avoided for physical reasons if a structure relaxation is requested.

x, y, z are real numbers (in Å) which specify the atomic position.

species\_name is a string descriptor which names the element on this atomic position; it must match with one of the species descriptions given in `control.in`.

This allows to place extra basis functions at specified locations outside the actual atoms, e.g., allowing for a counterpoise correction of basis set superposition errors.

---

**Tags for general section of control.in:**

---

**Tag: hydro\_cut** (control.in)Usage: `hydro_cut` flag

Purpose: Determines whether or not hydrogenic functions are subject to a numerical cutoff potential.

flag is a logical expression, either `.true.` or `.false.` Default: `.true.`

This tag should be kept at the default value unless for testing purposes (e.g., comparing to other codes).

---

**Tag: k\_grid** (control.in)Usage: `k_grid` n1 n2 n3

Purpose: Sets up an evenly split k-points grid along the reciprocal lattice vectors of a periodic calculation

n1, n2, n3 : integer numbers defining the number of k-point splits along the first, second and third reciprocal axis of the first Brillouin zone, respectively

*Note* that the order of n1, n2, n3 must *directly* correspond to the order in which the `lattice_vectors` are listed in `geometry.in`, through the definition and order of the reciprocal lattice. By default, the resulting `k_grid` is centered around the  $\Gamma$ -point, but can be shifted using the `k_offset` keyword below.

The keyword `symmetry_reduced_k_grid` now allows to make use of time-reversal symmetry to reduce the number of  $k$  points by a factor of nearly two. This option is the default.

---

**Tag: k\_offset** (control.in)Usage: `k_offset` f1 f2 f3Purpose: Defines a possible non- $\Gamma$  offset for the k-point grid in periodic boundary conditions.

f1, f2, f3 : Fractional coordinates (between zero and one) of a k-point offset , in units of the reciprocal lattice vectors. Default: (0., 0., 0.).

Can be used to shift the grid off- $\Gamma$  for better  $k$ -space sampling. For example, (0.5, 0.5, 0.5) together with even  $n_i$  for `k_grid` defines a Monkhorst-Pack [78] type grid. See Sec. 4.3 for details.

---

**Tag: k\_points\_external** (control.in)

Usage: `k_points_external`

Purpose: Instead of an internally specified  $k$ -point grid, allows to specify an externally read  $k$ -grid from a file `k_list.in`.

This option is useful to specify an uneven special  $k$ -point set [20], etc.

If specified, `k_points_external` expects a separate input file `k_list.in` to be provided in the same working directory as all other input files. The format of `k_list.in` is as follows:

```
line 1:      n1 n2 n3
line 2:      Nk
lines 3 - Nk+2: k1 k2 k3 weight
```

All lines are read as free format.  $n1$ ,  $n2$ ,  $n3$  are integers, specifying the descriptors of a 3D  $k$ -point grid for reference only.  $Nk$  is the (integer) number of  $k$ -points following in the file. For each  $k$ -point, a separate line is expected, including via  $k1$ ,  $k2$ ,  $k3$  the coordinates of this point in units of the reciprocal lattice vectors, and via *weight* the integration weight of this  $k$ -point in all Brillouin zone integrals.

**Tag:** `symmetry_reduced_k_grid` (`control.in`)

Usage: `symmetry_reduced_k_grid` flag

Purpose: Determines whether or not to make use of time-reversal symmetry.

flag is a logical expression, either `.true.` or `.false.` Default: `.true.`

**Tag:** `wave_threshold` (`control.in`)

Usage: `wave_threshold` threshold

Purpose: Determines the outer radius beyond which a radial function is considered zero.

threshold is a small positive number. Default:  $10^{-6}$ .

A radial function is considered zero (not evaluated) beyond the radius where  $u(r)$  and its first and second derivatives become smaller than `threshold`. The default is chosen such as to not affect any results at all.

**Tag:** `calculate_atom_bsse` (`control.in`)

Usage: `calculate_atom_bsse` flag

Purpose: Allows calculation of the basis set superposition error (BSSE) corrected atomization energy.

flag is a logical string, either `.false.` or `.true.` Default: `.false.`

The atomization BSSE correction for a molecular structure is defined as:

$$\Delta_{ac} = \sum_x [E^x(x) - E^x(sys)] \quad (3.7)$$

where  $E^x(x)$  is the energy of the atom  $x$  calculated using only its basis set and  $E^x(sys)$  is its energy calculated with the basis set of the whole structure. The BSSE corrected total energy is then

$$E^{BSSE} = E^{sys}(sys) + \Delta_{ac} \quad (3.8)$$

The atomization BSSE correction is usually small in the case of the LDA/GGA functionals, but can become significant for methods with explicit correlation like RPA and MP2.

The BSSE corrected atomization energy implemented here, on the other hand, is

$$E_{at}^{BSSE} = E^{sys}(sys) - \sum_x E^x(sys) \quad (3.9)$$

In the case where relative energies between different conformations of the same molecule are needed,  $(E_{at}^{BSSE})_{rel} \equiv (E^{BSSE})_{rel}$ .

If the calculation of the full system is performed without spin polarization, the total energy of each atom will also be calculated without spin polarization (and vice versa). In this case, specially when performing the *scf* cycle with HF, symmetry breaking of the electronic configuration of certain atoms may occur, which might lead to wrong conclusions. Support for constraining/breaking the symmetry of the atoms is under development.

The keyword is currently implemented for use with cluster geometries only. Both RPA and MP2 as the `total_energy_method` are supported.

## Subtags for *species* tag in `control.in`:

### **species sub-tag:** `basis_acc` (`control.in`)

Usage: `basis_acc` threshold

Purpose: Technical cutoff criterion for on-site orthonormalization of radial functions

threshold is a small positive real threshold. Default:  $10^{-4}$ .

Before any calculation, all radial functions for a single species are Gram-Schmidt orthonormalized. If the norm of the function after orthonormalization is smaller than threshold, that function is omitted.

### **species sub-tag:** `basis_dep_cutoff` (`control.in`)

Usage: `basis_dep_cutoff` threshold

Purpose: Basis function dependent adjustment of the confinement potential for this species

threshold is either a positive real number, or can be explicitly set `.false..`  
Default:  $10^{-4}$ .

If not `.false.`, the onset of the basis confining potential (see `cut_pot` tag below) is adjusted separately for each basis function, such that the norm of this basis function *outside*  $r_{\text{onset}}$  is smaller than threshold. The *maximum* possible onset radius is still given by the value explicitly specified by the `cut_pot` tag.

### **species sub-tag:** `confined` (`control.in`)

Usage: `confined` n l radius

Purpose: Adds a confined free-atom like radial function to the basis set.

n is the (integer) radial quantum number.

l is a character, specifying the angular momentum ( *s*, *p*, *d*, *f*, ...).

radius is the onset radius of the confining potential (in atomic units, 1 a.u. = 0.529177 Å). If the word `auto` is specified *instead* of a numerical value, the default onset radius given in the `cut_pot` tag is used.

The defining potential for this basis function type consists of the non-spinpolarized, self-consistent spherical free-atom potential (possibly itself confined, using the `cut_free_atom` tag), and a confining potential. The shape of the confining potential is the same for all basis functions of a given species, and set using the `cutoff_type` and `cut_pot` subtags.

### **species sub-tag:** `core` (`control.in`)

Usage: `core n l`

Purpose: Defines the top “core” shell of the species for this angular momentum.

`n` is the (integer) radial quantum number.

`l` is a character, specifying the angular momentum ( `s`, `p`, `d`, `f`, ...).

Currently not needed for production calculations, but listed here because the “core” infrastructure is currently being reworked and may see useful additions in the near future. This flag defines which electrons of the species are considered “core” electrons, and which enter as explicit valence electrons.

---

**species sub-tag:** `core_states` (control.in)

Usage: `core_states number`

Purpose: Independent determination of the number of states that are core states in the current species.

`number` is an integer number (to be multiplied by 2 for the number of core electrons). Default: 0.

*Experimental at present, not needed for any production purposes.* See also the `core` keyword. The `core_states` keyword should interact with the `core` keyword, but does not yet, since any associated functionality is still under development. Both keywords are listed here for future reference only.

---

**species sub-tag:** `cut_atomic_basis` (control.in)

Usage: `cut_atomic_basis flag`

Purpose: **Only** relevant to decide whether the `basis_dep_cutoff` keyword also applies to atomic-type (minimal) radial functions.

`flag` is a logical expression, either `.true.` or `.false.` Default: `.false.`

**This keyword applies only to the setting specified by keyword `basis_dep_cutoff`.** Do **not** enable it routinely without thorough testing.

By default, the minimal basis functions in FHI-aims are subject to the cutoff potential with the fixed onset specified by the `cut_pot` keyword. However, the more restrictive `basis_dep_cutoff` keyword does *not* apply to the minimal basis by default.

This can be changed by setting `cut_atomic_basis` to `.true.`, but the associated total energy changes are significantly larger than for other basis functions. By default, we therefore do not recommend adding a tighter cutoff to the minimal basis functions at this time. It is, however, possible that this effect is mainly a systematic error between the core states, and after further testing, we may yet choose to enable this keyword *if* we can guaranteed that its use is safe.

---

**species sub-tag:** `cut_pot` (control.in)

Usage: `cut_pot` onset width scale

Purpose: Specifies the numerical parameters for the general (default) confinement potential  $v_c(r)$  for all basis functions of this species.

`onset` specifies the default onset radius of the cutoff potential, in Å ( $v_c(r)=0$  for  $r < r_{\text{onset}}$ ).

`width` specifies the radial width  $w$  of the cutoff potential, in Å ( $v_c(r)=\infty$  for  $r > r_{\text{onset}} + w$ ).

`scale` is a scaling parameter to increase or decrease the numerical value of  $v_c$ .

This tag is mandatory, since it specifies `onset`, a critical parameter that allows to tune the efficiency of a calculation for a given target accuracy. Unless reduced by the `basis_dep_cutoff` tag, `onset` is the default onset radius used to construct all valence (minimal) and hydrogen-like basis functions of this species. In addition, any confined free-atom or free-ion like radial functions use this onset radius if `auto` is used in their specification.

Notes: The functional form of  $v_c(r)$  can be selected using the `cutoff_type` keyword, and `width` and `scale` apply to this shape. Modifying these latter parameters is usually not necessary for a production calculation, but the `onset` value should be verified at least as a quick numerical check.

**species sub-tag:** `cutoff_type` (`control.in`)

Usage: `cutoff_type` identifier

Purpose: Specifies the functional form of the confinement potential associated with this species.

`identifier` is a string that selects a given confinement potential shape as specified in the code. Default: `exp(1-x)_(1-x)2`.

All confinement potentials in FHI-aims are characterized by the rigorous boundaries  $v_c(r)=0$  for  $r < r_{\text{onset}}$  and  $v_c(r)=\infty$  for  $r > r_{\text{cut}} = r_{\text{onset}} + w$ , where  $r_{\text{onset}}$  may depend on the basis function, and  $w$  is the width specified by the `cut_pot` tag. In addition, each shape contains a scaling parameter  $s$ , also specified via the `cut_pot` tag.

Available confinement potential shapes (`identifier`) for  $r_{\text{onset}} < r < r_{\text{cut}} = r_{\text{onset}} + w$  are:

- `exp(1-x)_(1-x)2` :

$$v_c(r) = \exp\left(\frac{w}{r - r_{\text{onset}}}\right) \cdot \frac{1}{(r - r_{\text{cut}})^2}$$

(the default in FHI-aims)

- `junquera` :

$$v_c(r) = \exp\left(\frac{w}{r - r_{\text{onset}}}\right) \cdot \frac{1}{(r - r_{\text{cut}})}$$

(the form originally suggested by Junquera et al. [56])

- `x2_(1-x2)`

$$v_c(r) = (r - r_{\text{onset}})^2 \cdot \frac{1}{(r - r_{\text{cut}})^2}$$

**species sub-tag: gaussian** (control.in)

Usage: `gaussian` L N [alpha]

[ alpha\_1 coeff\_1 ]

[ alpha\_2 coeff\_2 ]

[ ... ]

[ alpha\_N coeff\_N ]

Purpose: Adds a Gaussian-based radial function to the basis set.

Restriction: This basis function type is not subject to a cutoff potential. It may therefore require a wider `radial_base` integration grid than the standard NAO's in FHI-aims.

L is an integer number, specifying the angular momentum

N is an integer number, specifying how many primitive Gaussians comprise the present radial function

alpha : If N=1, this is the exponent defining a primitive Gaussian function [in bohr<sup>-2</sup>].

alpha\_i coeff\_i : If N>1, i = 1, ..., N additional lines specify exponents  $\alpha_i$  and expansion coefficients  $g_i$  for a non-primitive linear combination of Gaussians.

FHI-aims allows to use Gaussian-based radial functions to compare to existing popular Gaussian-based implementations of quantum chemistry. These functions can either be *primitive* Gaussians,

$$u(r) = \frac{1}{Norm} r^{L+1} \cdot \exp(-\alpha r^2), \quad (3.10)$$

or *non-primitive* linear combinations.

$$u(r) = \frac{1}{Norm} r^{L+1} \cdot \sum_{i=1}^{i=N} g_i \exp(-\alpha_i r^2). \quad (3.11)$$

In existing quantum chemistry codes, Gaussian basis functions can be defined either as *spherical* Gaussians [Eq. (3.10) above], or as *cartesian* Gaussians,

$$\phi(\mathbf{r}) = x^k y^m z^n \exp(-\alpha r^2), \quad \text{where } k + m + n = L. \quad (3.12)$$

This behavior can be mimicked using the `pure_gauss` tag (see below). Finally, note that in order to use an *exclusively* Gaussian-based basis set, you must prevent the use of the minimal free-atom like NAO basis functions using the `include_min_basis` tag.

**species sub-tag: hydro** (control.in)

Usage: `hydro n l z_eff`

Purpose: Adds a hydrogen-like radial function to the basis set.

`n` is the (integer) radial quantum number.

`l` is a character, specifying the angular momentum ( `s`, `p`, `d`, `f`, ...).

`z_eff` scales the radial function as an effective nuclear charge in the defining Coulomb potential  $z_{\text{eff}}/r$ .

By default, hydrogen-like basis functions in FHI-aims are subject to the numerical confinement potential given by `cutoff_type` and `cut_pot`. Optionally, analytical hydrogen-like functions (no confinement) can be requested using the global `hydro_cut` tag.

**species sub-tag:** `include_min_basis` (control.in)

Usage: `include_min_basis` flag

Purpose: Allows to exclude the minimal basis of free-atom valence functions from the basis set.

flag is a logical expression, either `.true.` or `.false.` Default: `.true.`

This flag is normally only useful to compare explicitly with basis sets from other methods, usually Gaussian basis sets.

**species sub-tag:** `ion_occ` (control.in)

Usage: `ion_occ n l occupation`

Purpose: Specifies the shell occupation of a radially symmetric, non-spinpolarized free ion that defines any ionic basis functions.

Restriction: Only one type of ion can be used to define ionic basis functions for a given species.

`n` is the (integer) radial quantum number.

`l` is a character, specifying the angular momentum ( `s`, `p`, `d`, `f`, ...).

`occupation` is the number of electrons in the topmost occupied valence shell of this ion.

This tag defines an ionic configuration, but does not actually add any ionic functions to the basis used in the present calculation. Actual basis functions are added by the `ionic` keyword.

Only the topmost valence shell of each angular momentum channel of the ion is specified. All lower-lying shells are assumed to be completely filled for the self-consistent spherical free-ion calculation.

**species sub-tag:** `ionic` (control.in)

Usage: `ionic n l radius`

Purpose: Adds a free-ion like radial function to the basis set.

`n` is the (integer) radial quantum number.

`l` is a character, specifying the angular momentum ( `s`, `p`, `d`, `f`, ...).

`radius` is the onset radius of the confining potential (in atomic units, 1 a.u. = 0.529177 Å). If the word `auto` is specified *instead* of a numerical value, the default onset radius given in the `cut_pot` tag is used.

**species sub-tag:** `pure_gauss` (control.in)

Usage: `pure_gauss flag`

Purpose: If `.true.`, any `gaussian` basis functions for this species will be purely spherical Gaussians.

`flag` is a logical string, either `.false.` or `.true.` Default: `.true.`

See keyword `gaussian` for the distinction between spherical and cartesian Gaussian functions. In short, cartesian and spherical Gaussian functions are equivalent *except* that for a given  $L$ , cartesian Gaussians add a so-called angular momentum contamination. If `pure_gauss` is specified, this angular momentum contamination is mimicked by FHI-aims.

Consider the simple (textbook!) three-dimensional harmonic oscillator in quantum mechanics. This can be solved either in cartesian coordinates, or in spherical coordinates. If solves in cartesian coordinates, you will find that there are six degenerate solutions for the principal quantum number 2, five of which correspond to  $l=2$  ( $d$  channel), but one of which corresponds to  $l=0$  ( $s$  channel). This is the exact angular momentum contamination exhibited by the cartesian definition of a Gaussian basis function.

**species sub-tag:** `valence` (control.in)

Usage: `valence n l occupation`

Purpose: Specifies the shell occupation of the radially symmetric, non-spinpolarized free atom that defines the minimal basis.

`n` is the (integer) radial quantum number.

`l` is a character, specifying the angular momentum ( `s`, `p`, `d`, `f`, ...).

`occupation` is the number of electrons in the topmost occupied valence shell of this ion.

Only the topmost valence shell of each angular momentum channel of the atom is specified. All lower-lying shells are assumed to be completely filled for the self-consistent spherical free-atom calculation. The valence occupation must be defined explicitly for each `species`.

The self-consistent free-atom potential generated by this calculation is used to generate all minimal and `confined` basis functions used for this species, after the confining

potential is added.

The self-consistent free-atom calculation can itself be confined by a different confining potential, the onset of which is specified by the `cut_free_atom` keyword.

For DFT-LDA/GGA, the same `xc` functional that is used in the full three-dimensional calculation is also used to define the self-consistent free atom. For any methods involving Hartree-Fock exchange (e.g., hybrid functionals), the free atom is generated using the `pw-lda` LDA functional.

The self-consistent free atom density generated here is also used in the construction of partition functions for the Hamiltonian integrals and the Hartree potential, as well as to build the initial charge density (unless otherwise requested!) and the reference charge density subtracted before constructing the Hartree potential.

## 3.5 Integration, grids, and partitioning

The next single most important set of specifications required for FHI-aims are the settings regarding the numerical grids used in many contexts. Details regarding the shape and physical motivation behind these grids are given in Refs. [12, 44], and we do not repeat them here.

Notice that the actual required grids may depend on the context of the calculation, for example whether Hartree-Fock, hybrid functionals, and or *GW* calculations are required. In these cases, some specific settings may require tightening, and some defaults may automatically be chosen differently depending on whether or not those techniques are used.

Specifically, the present section deals with the following topics:

- the 1D logarithmic grid infrastructure required for atomic / free-atom like calculation
- radial and angular grids for all three-dimensional integrals
- shaping the partition functions used to split the full three-dimensional integrals into effective atom-per-atom pieces
- Splitting the grids into different batches for localization / parallelization efficiency

While many of the settings below take safe defaults for standard FHI-aims calculations and need not be modified, it is particularly important to verify the accuracy and efficiency of all three-dimensional integration grids ([radial\\_base](#), [angular\\_grids](#), and associated tags), since these determine the performance of the code. In the `species_defaults` files, (very) safe settings for DFT-LDA/GGA are provided, but for many tasks, may be reduced at very little accuracy loss.

## Tags for general section of `control.in`:

---

### Tag: `batch_size_limit` (`control.in`)

Usage: `batch_size_limit` value

Purpose: Hard upper bound to the number of points in an integration batch.

Restriction: Applies to the `maxmin` and `octree` `grid_partitioning_method`.  
value is an integer number. Default: 200.

See `grid_partitioning_method` and Ref. [44] for details regarding integration batches.

---

### Tag: `force_lebedev` (`control.in`)

Usage: `force_lebedev` type

Purpose: Allows to switch between Delley's [23] angular grids (17 digits) and the original angular grids tabulated by Lebedev and Laikov [66, 67, 68] (12 digits)

type is a keyword (string), either `original` or `Delley`. Default: `Delley`.

This option need not be changed (or invoked) in any normal runs, since there is no quantitative difference between integrals with Delley's and Lebedev's tabulated grids to our knowledge.

Lebedev's grids may be explicitly invoked when denser angular grids than 1202 points (already very dense!) per radial integration shell around each species are required. In detail, grids with the following numbers of grid points are provided:

- Delley : 6, 14, 26, 50, 110, 194, 302, 434, 590, 770, 974, 1202
- Lebedev : 6, 14, 26, 38, 50, 86, 110, 146, 170, 194, 302, 350, 434, 590, 770, 974, 1202, 1454, 1730, 2030, 2354, 2702, 3074, 3470, 3890, 4334, 4802, 5294, 5810

These numbers of grid points can be invoked in the subtags of the `angular_grids` specified description for fixed angular grids (the default in the preconstructed `species_defaults` files), and in further tags such as `angular` or `angular_min`.

---

### Tag: `grid_partitioning_method` (`control.in`)

Usage: `grid_partitioning_method` method

Purpose: Allows to switch between different methods to partition the full (3D) integration grids into batches for individual operations.

method is a string, characterizing one of the different methods outlined below.

Default: `serial-maxmin`, `parallel-parallel_hash+maxmin`

Partitioning the integration grid properly can be performance-critical for the expensive

grid-based Hamiltonian integration and charge density update steps. Details on these methods are given in Ref. [44]. In particular, we support:

- `maxmin` : The default for serial computations: the “grid-adapted cut-plane” method described in Ref. [44]
- `parallel_hash+maxmin` : The default for all parallel runs. We first hash the grid points to tasks by the geometric location and then run a `maxmin` algorithm locally in each task.
- `parallel_maxmin` : Memory-parallel implementation of the `maxmin` method. However, the exact implementations requires rather much communication, and has been superseded by `parallel_hash+maxmin`.
- `octree` : The “octree” method described in Ref. [44]
- `parallel_octree` Parallel version of the “octree” method described in Ref. [44]. Only useful for research purposes—superseded by `parallel_hash+maxmin` for all practical applications.
- `octant` Simple partitioning of the grid into “octants” of each radial integration shell around each atom.

Note that additional parameters may be invoked to specify details for these methods, most importantly `batch_size_limit` and `points_in_batch`.

We mention for completeness that FHI-aims supports further, experimental grid batching methods, including the possibility to link to external libraries. The associated method strings are `tetgen+metis`, `qhull+metis`, `nearest+metis`, and `group`. As discussed in Ref. [44], the conceptually simpler `maxmin` method performs as well or even better than these “bottom-up” type approaches, and should be preferred.

---

**Tag:** `min_batch_size` (`control.in`)

Usage: `min_batch_size` value

Purpose: Sets the minimum number of points allowed in an integration batch.

Restriction: Affects the octree `grid_partitioning_method` method only.  
value is an integer number. Default: 1.

No need to tweak for standard production calculations. See `grid_partitioning_method` for details regarding integration batches.

---

**Tag:** `partition_acc` (`control.in`)

Usage: `partition_acc` threshold

Purpose: If the partition function norm for 3D integrals and the Hartree potential is below threshold, that integration point is ignored.

threshold is a small positive real number. Default:  $10^{-15}$ .

See Ref. [12] for details regarding “partitioning of unity” of the charge density in integrations and the Hartree potential. The partition functions  $p_{\text{at}}(\mathbf{r})$  are only calculated if their denominator (the norm; e.g.  $\sum_{\text{at}} \rho_{\text{at}}/|\mathbf{r} - \mathbf{R}_{\text{at}}|^2$ ) is greater than value, else that integration point is ignored.

Notice that this type of partitioning is strictly rigorous for integrands that extend no further than the free-atom like densities used to define our partition functions. This is always true for DFT-LDA/GGA, with NAO’s but if you suspect (e.g., with very diffuse Gaussian basis functions) some kind of integration noise, reducing threshold may be a good first test.

**Tag:** `partition_type` (control.in)

Usage: `partition_type` type

Purpose: Specifies which kind of partition table is used for all three-dimensional integrations.

type : A string that specifies which kind of partition table is used. Default: `stratmann_smoother`

Usually, this tag need not be modified from the default. See the Computer Physics Communications description of FHI-aims for a description of the numerical integration technique used in FHI-aims.

In brief, each extended three-dimensional integrand is broken down into atom-centered pieces, using a set of localized, atom-centered partition functions:

$$p_{\text{at}}(\mathbf{r}) = \frac{g_{\text{at}}(\mathbf{r})}{\sum_{\text{at}'} g_{\text{at}'}(\mathbf{r})}. \quad (3.13)$$

where  $g_{\text{at}}(\mathbf{r})$  is an atom-centered weight function. The following options for type are available:

- `rho_r2` :  $g_{\text{at}}(\mathbf{r}) = n_{\text{at}}^{\text{free}}(r)/r^2$   
(first suggested by Delley [22]).
- `rho_r` :  $g_{\text{at}}(\mathbf{r}) = n_{\text{at}}^{\text{free}}(r)/r$
- `rho` :  $g_{\text{at}}(\mathbf{r}) = n_{\text{at}}^{\text{free}}(r)$
- `fermi` : *Deprecated—do not use.* A Fermi-function like approach, requires two additional parameters.
- `stratmann`: The shape suggested by Stratmann *et al.*, Ref. [103]. This saves  $\approx 10$ -20 % of the numerical effort compared to `rho_r2`. More importantly, however, our recent testing shows that `stratmann` is also significantly accurate in some corner cases where the effects of integration accuracy even make a difference. *Note the properly bounded “stratmann\_smoother” default function below. Straight “stratmann” should not be used.*
- `stratmann_smooth`: Partial update to guarantee a smooth edge at the “outer radius” or atoms.

`stratmann_smoother`: Corrected version of the stratmann partition table. The following explanation refers to the prescription given in Eqs. (8), (11), and (14) of Ref. [103]. The actual (normalized) partition function is given by Eq. (10). At each grid point, it depends on a product of cell functions Eq. (9) over potentially all atoms in the system—unless its cell function is equal to one, a faraway atom may contribute to the partition function at a given grid point. Through the definition of  $\mu_{ik}$  in Eq. (4) of Ref. [103] and the limitation of the cell function to unity for  $\mu_{ik} < a = 0.64$  through Eqs. (11) / (14), the distance from which an atom can contribute is restricted, but potentially to a very large radius indeed. This becomes a problem for periodic systems (in our case, a theoretical radius of 25 Å would have resulted even for light settings and the farthest grid points from each atom, set to 5 Å for light settings).

To avoid an overly large volume of contributing atoms, we restrict the list of contributing atoms to only those whose free-atom charge density would not be zero at the integration point in question. To that end, Eq. (8) of Ref. [103] is additionally multiplied with a function that smoothly interpolates between the original  $s$  of Stratmann and coworkers and unity. The interpolation is done only for atom distances between 0.8 and 1.0 times their free-atom radius, and uses a  $[1 - \cos(2x)]$ -like interpolating function. The bottom line is that we get the benefits of both the Stratmann table and a restricted atom list without any discontinuities or wiggles as a function of atomic positions or unit cell vectors — which is as it should be.

Note that the free atom electron density  $n_{\text{at}}^{\text{free}}(r)$  still determines the extent of many partition function types. This is controlled by the `cut_free_atom` keyword. See also the `hartree_partition_type` keyword, which presently must have the same setting as the `partition_type` keyword.

---

**Tag:** `points_in_batch` (`control.in`)

Usage: `points_in_batch` value

Purpose: Target number of grid points per integration batch.

Restriction: Applies to the `maxmin` and `octree` `grid_partitioning_method`.  
value is an integer number. Default: 100.

See `grid_partitioning_method` and Ref. [44] for details regarding integration batches.

## Subtags for *species* tag in `control.in`:

---

### **species sub-tag:** `angular` (`control.in`)

Usage: `angular` `limit`

Purpose: For *self-adapting* angular integration grids, the maximum allowed number of points per radial shell.

Restriction: This flag has no effect for species where `angular_grids` is explicitly specified (the default in our `species_default` files).

`limit` is the maximum allowed number of integration points per radial shell.

This option is only meaningful for self-adapting angular grids, which are *not* the recommended default for production calculations with FHI-aims – (i) because these grids are often rather dense, and (ii) because they are meaningful only for cluster-type geometries. In order to specify self-adapting angular grids anyway, you must also set the keywords `angular_min` and `angular_acc`.

The available values of integration points in given angular grids are listed with the keyword `force_lebedev`.

---

### **species sub-tag:** `angular_acc` (`control.in`)

Usage: `angular_acc` `threshold`

Purpose: For *self-adapting* angular integration grids, specifies the desired integration accuracy for the initial Hamiltonian and overlap matrix elements.

Restriction: Use only for cluster-type geometries.

`threshold` is a small positive real number; if 0., no adaptation is performed.  
Default: 0.

If `threshold` is not zero, this option invokes the self-adaptation of all angular integration grids, within the limits given by `angular_min` and `angular`. The adaption criteria are the initial Hamiltonian / overlap matrix integrals.

In all preconstructed `species_default` files, we specify reliable angular integration grids for all elements for DFT. No adaptation is required. For the curious, our own grids are adapted for symmetric dimers at a tight bond distance, using `threshold = 10-8`.

---

### **species sub-tag:** `angular_grids` (`control.in`)

Usage: `angular_grids` `method`

Purpose: Indicates how the angular integration grids (in each radial integration shell) for this `species` are determined.

`method` is a string, either `auto` or `specified`.

The standard `species_default` files provided with FHI-aims provide specified an-

gular grids (on the safe side, i.e., rather dense) for each `radial_base` integration shell around an atom. The line:

`angular_grids` specified

must be immediately followed by a series of lines with

`division` [...]

`outer_grid` [...]

tag(s). These contain the actual grid specification.

If method `auto` is given, appropriate specifications for self-adapting grids should be included in `control.in` (keywords `angular`, `angular_min`, `angular_acc`).

#### **species sub-tag: `angular_min` (`control.in`)**

Usage: `angular_min` value

Purpose: specifies the minimum number of angular grid points per radial integration shell

value is the minimum number of grid points per shell.

For specified `angular_grids`, acts as a lower bound for the number of points per radial shell (specified grids will be increased accordingly).

For self-adapting angular grids, use together with the `angular` and `angular_acc` keywords.

In practice, value will be reduced to the next-highest available Lebedev integration grid (see `force_lebedev` tag for possible values).

#### **species sub-tag: `cut_free_atom` (`control.in`)**

Usage: `cut_free_atom` type [radius]

Purpose: Adds a cutoff potential to the initial, non-spinpolarized free-atom calculation that yields free-atom densities and potentials for many basic tasks.

type : A string, either `finite` or `infinite`. Default: `finite` for DFT-LDA/GGA; `infinite` for Hartree-Fock, hybrid functionals, *GW*, etc.

radius : A real number, in Å: Onset radius for the cutoff potential, as defined in the `cut_pot` tag. Default: For DFT-LDA/GGA,  $r_{\text{onset}}$  as given by the `onset` parameter in `cut_pot` .

Although this is a technical parameter (ideally, no influence on self-consistent, converged results), it has important implications for a variety of numerical tasks in the code:

- It influences (slightly) the basis-defining potential for the minimal basis, and for `confined` basis functions.
- It limits the radius of the free-atom density, which in turn limits the extent of the default integration partition table. For DFT-LDA/GGA, this extent need must not be smaller than the radius of the most extended basis function, but it also need

not be larger, since all integrands are zero outside anyway. This is *not* the case for the two-electron Coulomb operator, which is needed for Hartree-Fock, hybrid functionals, *GW*, etc, in which case the default is currently *infinite* (no cutoff potential applied).

- It also limits the extent of the partition table used for the Hartree potential. Especially in periodic calculations, it is vital that the real-space part of the Hartree potential is kept small. In that case, it is thus critical to keep *radius* as small as possible.

Usually, the default specified in the code should be accurate for all requirements. If, however, you suspect some kind of integration noise which is not related to the grid, increasing the `cut_free_atom` value may be a good test.

#### **species sub-tag: division** (control.in)

Usage: `division` radius points

Purpose: For specified `angular_grids`, the number of angular points on all radial shells that are within *radius*, but not within another, *smaller* division.

Restrictions: Meaningful only in a block immediately following an `angular_grids` specified line.

`radius` : Outer radius (in Å) of this division.

`points` : Integer number of angular points requested in this division (see `force_lebedev` tag for possible values).

Use the `outer_grid` tag to specify the number of angular grid points used outside the outermost division radius.

#### **species sub-tag: innermost\_max** (control.in)

Usage: `innermost_max` number

Purpose: Monitors the quality of the radial integration grid.

`number` is an integer number, corresponding to a radial grid shell. Default: 4.

If, after on-site orthonormalization, a radial function's innermost extremum is inside the radial grid shell `number`, counting from the nucleus, that radial function is rejected in order to prevent inaccurate integrations.

#### **species sub-tag: logarithmic** (control.in)

Usage: `logarithmic` `r_min` `r_max` `increment`

Purpose: Defines the dense one-dimensional “logarithmic” grid for the direct solution of all radial equations (free atom quantities, Hartree potential).

`r_min` is a real number (in bohr); the innermost point of the logarithmic grid is defined as  $r(1)=r\_min/Z$ , where  $Z$  is the atomic number of the `nucleus` of the `species`. Default: 0.0001 bohr.

`r_max` is a real number (in bohr), the outermost point of the logarithmic grid,  $r(N)$ . Default: 100 bohr.

`increment` is a real number, the increment factor  $\alpha$  between successive grid points,  $r(i) = \alpha \cdot r(i - 1)$ . Default: 1.0123.

The number of logarithmic grid shells,  $N$ , is uniquely determined by `r_min`, `r_max`, and `increment`. Specifying a dense logarithmic grid is not performance-critical.

**species sub-tag:** `outer_grid` (`control.in`)

Usage: `outer_grid` `points`

Purpose: For specified `angular_grids`, the number of angular points on all radial shells outside the largest `division`.

Restrictions: Meaningful only in a block immediately following an `angular_grids` specified line.

`points` : Integer number of angular points (see `force_lebedev` tag for possible values).

Use the `division` tag to specify the number of angular grid points used for radial shells within specified radii.

**species sub-tag:** `radial_base` (`control.in`)

Usage: `radial_base` `number` `radius`

Purpose: Defines the basic grid of radial integration shells according to Ref. [7]

`number` is an integer number (the total number of grid points,  $N$ ).

`radius` is a positive real number which specifies the outermost shell of the basic grid,  $r_{\text{outer}}$ , in Å.

The location of the number radial shells is given by

$$r(i) = r_{\text{outer}} \cdot \frac{\log\{1 - [i/(N + 1)]^2\}}{\log\{1 - [N/(N + 1)]^2\}} \quad (3.14)$$

With this prescription, shell  $i=0$  would be located exactly at  $r(i) = 0$ , and shell  $i=N + 1$  would be located exactly at  $r(i) = \infty$ , i.e., this provides an exact mapping of the interval  $[0, \infty]$ .

The FHI-aims `species_default` files provide values for `number` according to the formula  $N=1.2 \times 14(\text{nucleus}+2)^{1/3}$ , as determined empirically in Ref. [7]. These “basic” grids are can then be augmented by adding uniform subdivisions, using the `radial_multiplier`

keyword described below.

---

**species sub-tag:** `radial_multiplier` (control.in)

Usage: `radial_multiplier` number

Purpose: Systematically increases the radial integration grid density.

value is an integer, number specifying the number of added subdivisions per basic grid spacing. Default: value = 2

The basic grid of  $N$  radial shells (see `radial_base` definition) is evenly subdivided `number-1` times, to yield  $\text{number} \cdot (N + 1) - 1$  shells in the *actually used* integration grid. Thus, the `radial_multiplier` tag allows to systematically increase the number of radial shells (by factors). For example, `number=2` (the default) would yield  $2N + 1$  shells total.

Note that some all-electron Gaussian basis sets contain either very high or very low exponents. If such basis sets are used for test purposes, it may be necessary to test the convergence of the radial integration grid directly by increasing the `radial_multiplier`.

## 3.6 Electron density update

In FHI-aims, the first step of a new iteration is the update of the electron density based on the output Kohn-Sham orbitals produced by a previous step.

The present section covers only the *actual* density update. Techniques relevant for the self-consistent *convergence* of the whole calculation (electron density mixing, preconditioning, etc.) are covered separately in Sec. [3.10](#).

## Tags for general section of `control.in`:

---

**Tag:** `density_update_method` (`control.in`)

Usage: `density_update_method` type

Purpose: Governs the selection of the density update type.

Restriction: For periodic boundary conditions, only the density-matrix based electron density update is supported.

Default: Cluster case: `automatic`. Periodic case: `density_matrix`

Choices for type:

- `orbital` : Use Kohn-Sham orbitals based update
- `density_matrix` : Use density-matrix based update method. Required for periodic systems.
- `automatic` : Selects the best update method automatically, based on the expected amount of work.
- `split_update_methods` : Charge density is updated via Kohn-Sham orbitals and force is updated via density-matrix

If not specified, default for cluster geometries is the automatic selection of the density update method.

See Ref. [12] for details regarding density update mechanisms. In general, FHI-aims offers an electron density update based on Kohn-Sham orbitals [ $O(N^2)$  with system size, but faster for finite systems up to  $\approx 100$ -500 atoms depending on basis size], and an  $O(N)$  rewrite based on the density matrix. This should be used for large systems, and is the default for periodic systems.

For the non-periodic case, the current code version determines the switching point between the orbital-based update and the density-matrix based update automatically through some heuristics. This procedure guarantees that accidental  $O(N^2)$  calculations will not happen for very large systems, but the optimum cross-over point may not always be exactly found. If you are planning long runs of essentially the same geometry (molecular dynamics trajectories are a good example), you may save some time by performing some explicit benchmarks first. You can then specify the optimum density update method for *your* own case, instead of relying on our heuristics.

## 3.7 Electrostatic (Hartree) potential

This section describes the method used to compute the electrostatic (Hartree) potential in FHI-aims. For a more exhaustive description, please refer to Ref. [12].

Some central equations are repeated here in detail since, as a result, the calculation of the Hartree potential can be heavily customized by many analytically available accuracy / cutoff thresholds, given below.

For production calculations, it is important to note that *our standard accuracy thresholds in the Hartree potential are numerically sound, and usually do not require an explicit customization*. The only parameter which should be explicitly set is the angular momentum up to which the atom-centered partitioned charge density is expanded, `l_hartree` below.

As pointed out in Ref. [12], our experience is that energy differences are usually well converged for  $l_{\text{hartree}}=4$ , and total energy convergence at the level of a few meV is reached at  $l_{\text{hartree}}=6$ . Only in exceptional cases should different settings be required.

At the beginning of a calculation, we first compute the electrostatic potential associated with the initial superposition of free-atom densities,  $\sum_{\text{at}} n_{\text{at}}^{\text{free}}(|\mathbf{r} - \mathbf{R}_{\text{at}}|)$ :

$$v^{\text{es,free}}(\mathbf{r}) = \sum_{\text{at}} v_{\text{at}}^{\text{es,free}}(|\mathbf{r} - \mathbf{R}_{\text{at}}|) \quad (3.15)$$

This is the largest part of the Hartree potential, but is always accurately known from the solution of spherical free atoms on a dense logarithmic grid.

For a given electron density  $n(\mathbf{r})$  during the s.c.f. cycle, we then only ever compute the electrostatic potential associated with the *difference* to the superposition of free atoms,  $\delta v_{\text{es}}(\mathbf{r})$ , based on

$$\delta n(\mathbf{r}) = n(\mathbf{r}) - \sum_{\text{at}} n_{\text{at}}^{\text{free}}(|\mathbf{r} - \mathbf{R}_{\text{at}}|) \quad (3.16)$$

$\delta n(\mathbf{r})$  is first split up into a sum of *partitioned, atom-centered* charge multipoles,

$$\delta \tilde{n}_{\text{at},lm}(r) = \int_{r=|\mathbf{r}-\mathbf{R}_{\text{at}}|} d^2\Omega_{\text{at}} p_{\text{at}}(\mathbf{r}) \cdot \delta n(\mathbf{r}) \cdot Y_{lm}(\Omega_{\text{at}}) \quad (3.17)$$

(the sum of all partition functions at every point is always unity). Due to the finite extent of  $\delta n(\mathbf{r})$  and  $p_{\text{at}}(\mathbf{r})$  (both are controlled by the `cut_free_atom` keyword), the range of each component  $\delta \tilde{n}_{\text{at},lm}(r)$  is also bounded.

The Hartree potential components  $\delta \tilde{v}_{\text{at},lm}(r)$  are then determined on a dense, one-dimensional logarithmic grid, using classical electrostatics. The resulting  $\delta \tilde{v}_{\text{at},lm}(r)$  are then numerically tabulated, and evaluated elsewhere using cubic spline interpolation.

For *cluster* systems, it is important to note that the finite extent of  $\delta \tilde{n}_{\text{at},lm}(r)$  implies that the numerically tabulated part of  $\delta \tilde{v}_{\text{at},lm}(r)$  can also be kept finite. Outside this “multipole radius”,  $\delta \tilde{n}_{\text{at},lm}(r)=0$ , and  $\delta \tilde{v}_{\text{at},lm}(r)$  falls off analytically as

$$\delta \tilde{v}_{\text{at},lm}(r) = m_{\text{at},lm}/r^{l+1} \quad . \quad (3.18)$$

Instead of a spline evaluation, faraway atoms can thus be analytically accounted for using tabulated, constant *multipole moments*  $m_{\text{at},lm}$ . High- $l$  components can be analytically cut off as they approach zero at large distances.

In this approach, the effort to create the complete Hartree potential on the entire grid is determined by tabulating the contribution from *every* atom on *every* grid point,

$$v_{\text{es}}(\mathbf{r}) = v^{\text{es,free}}(\mathbf{r}) + \sum_{\text{at},lm} \delta\tilde{v}_{\text{at},lm}(|\mathbf{r} - \mathbf{R}_{\text{at}}|)Y_{lm}(\Omega_{\text{at}}) \quad . \quad (3.19)$$

The scaling is thus close to  $O(N^2)$  with system size, albeit reduced by high- $l$  multipole components falling off towards large distances.

For *periodic* systems, essentially the same equations hold, except that the Hartree potentials associated with the atom-centered charge densities  $\delta\tilde{n}_{\text{at},lm}(r)$  are here additionally split into a short-ranged real-space part, and a smooth, long-ranged reciprocal-space part (Ewald's method), by splitting

$$\frac{1}{r} = \frac{\text{erf}(r/r_0) + \text{erfc}(r/r_0)}{r} \quad (3.20)$$

(and similar for components of higher angular momentum). The summation of long-range tails thus happens in reciprocal space, using Fourier transforms. As a result, the scaling of this effort is no longer  $O(N^2)$ , but rather approaches  $O(N\ln N)$  in Fourier transforms.

The cutoff reciprocal space momentum for the Fourier part of the electrostatic potential,  $|\mathbf{G}_{\text{max}}|$ , is estimated using a small threshold parameter  $\eta$ :

$$|\mathbf{G}_{\text{max}}|_{\text{max}}^{\text{es}-2} \cdot \frac{1}{\mathbf{G}_{\text{max}}^2} \cdot \exp\left(-\frac{r_0^2 \mathbf{G}_{\text{max}}^2}{4}\right) = \frac{\eta}{10 \cdot 4\pi} \quad (3.21)$$

Our default choice for  $\eta$  (in atomic units, i.e., those used internally in the code) is  $\eta = 5 \cdot 10^{-7}$ , but this is somewhat overconverged, and a larger threshold value is probably sufficient for most situations. *Note* that Eq. (3.21) is slightly modified compared to the version given in Ref. [12].

### 3.7.1 Non-periodic Ewald method

For large, finite systems (more than 200 atoms) it is possible to use the so-called 'non-periodic Ewald method' in aims (keyword `use_hartree_non_periodic_ewald`). The basic idea of this method is to use interpolation to reduce the effort for calculating the Hartree term. Specifically, the method consists in computing the electrostatic potential not on the fine interpolation grid points but firstly on a coarse Cartesian grid. Subsequently, the values of the potential on the coarse grid are interpolated to the fine integration grid. If the Cartesian grid is sufficiently coarse, time is saved because of the reduced number of potential computations.

We use an evenly spaced, Cartesian grid with a certain grid width. Due to this fixed grid width, special attention has to be paid to the near-atom regions where the electron

density and hence also the potential oscillates strongly. This problem can be solved by using the Ewald decomposition which was originally developed for periodic systems. Ewald's method aims at separating large and small scales by adding and subtracting charge spheres with Gaussian radial shape to a lattice of monopoles. In terms of the potential, this yields  $\bar{q}/r = [\bar{q}/r - \Omega(r)] + \Omega(r)$  for each monopole, where  $\bar{q} := q/(4\pi\epsilon_0)$  and  $q$  is the monopole charge. The function  $\Omega(r) = \bar{q} \operatorname{erf}(r/r_0)/r$  is the potential of a Gaussian charge sphere with width parameter  $r_0$ . The first part of the decomposition  $\bar{q}/r - \Omega(r)$  decays quickly with increasing  $r$  so that this part is calculated in real space, while the second part  $\Omega(r)$  decays quickly in Fourier space so that it is calculated there. The two parts are often referred to as 'short range' and 'long range' part. However, this is somewhat misleading because the second part is actually defined in whole space. For this reason, we call the first part 'localized' and the second part 'extended'.

We can translate the classical Ewald decomposition to our case of a finite system by calculating the smooth extended part  $\Omega(r)$  on the coarse Cartesian grid, with subsequent interpolation to the fine integration grid points. In addition, we have to calculate the localized part in the vicinity of the nuclei where we cannot save any computational time [actually some time is lost since we have to compute  $\Omega(r)$  there, too].

In the classical Ewald method, Gaussian spheres are an excellent choice as auxiliary charges due to the quick convergence of both the localized part in real space and the extended part in Fourier space. However in our case, where we interpolate in real space, Gaussian spheres are not necessarily a proper choice. Therefore optimized charge distributions obtained from a variational method by W. Jürgens are used.

In order to reduce the number of grid points, we allow the Cartesian grid to have arbitrary orientation. More specifically, we are looking for a rectangular cuboid that covers all integration grid points but with minimum volume. This problem is solved approximately by using a common procedure that is based on principle component analysis.

## Tags for general section of `control.in`:

### Tag: `adaptive_hartree_radius_th` (`control.in`)

Usage: `adaptive_hartree_radius_th` threshold

Purpose: Determines the distance beyond which an analytical component  $\delta\tilde{v}_{\text{at},lm}(r)$  of the *periodic* (Ewald!) real-space Hartree potential for a given atom is considered zero.

threshold is a small positive real number. Default:  $10^{-8}$ .

Usually, this tag need not be modified from the default. Long-range multipole components  $\delta\tilde{v}_{\text{at},lm}(r)$  of the real-space (Ewald!) Hartree potential are not evaluated for distances where  $\delta\tilde{v}_{\text{at},lm}(r) < \text{threshold}$ . This tag provides similar functionality as the `multipole_threshold` tag for the cluster case (numerically different due to the absence of  $\text{erf}(r/r_0)$  in the cluster case).

### Tag: `distribute_leftover_charge` (`control.in`)

Usage: `distribute_leftover_charge` `.true./false`. Mostly relevant for surfaces, in particular when calculating electrostatic potentials, interface dipoles, or work functions. Due to the finite summation in the multipole expansion, the electronic and the nuclear charge may not cancel each other exactly. The remaining, apparent charge of the system is plotted in the output as *RMS charge density error from multipole expansion: (value)* and causes a slightly curved potential in vacuum region. As a result, e.g., the energy of the vacuum level depends on its position (which can be changed using the tag `set_vacuum_level`). If set to `.true.`, this keyword distributes the apparent charge evenly among all atoms in the system, causing a perfectly flat potential in the vacuum region.

### Tag: `hartree_convergence_parameter` (`control.in`)

Usage: `hartree_convergence_parameter` value

Purpose: Governs the Ewald-type short-range / long-range splitting of the Coulomb potential in Eq. (3.20).

value : Distance parameter  $r_0$  in Eq. (3.20) (in bohr). Default: 3.0 bohr.

Necessary for periodic boundary conditions only. May be changed from the default, but should not be set too small or too large (the compensating Gaussian charge density of the Ewald method must cancel the actual charge outside a radius that is still inside the partition table / integration grid for every atom.)

### Tag: `hartree_fp_function_splines` (`control.in`)

Usage: `hartree_fp_function_splines` `.true.` / `.false.`

Purpose: Switches on the splining of the Greens functions for the long-range Hartree multipole decomposition in periodic boundary conditions. This accelerates the calculation of the Hartree potential in large unit cells.

Default: `.true.`

**Tag:** `hartree_fourier_part_th` (`control.in`)

Usage: `hartree_fourier_part_th` `threshold`

Purpose: *Implicitly* determines the required reciprocal space cutoff momentum  $|\mathbf{G}_{\max}|$  for the Fourier summation of the long-range electrostatic potential (Ewald).

`threshold` is a real positive small number [ $\eta$  in Eq. (3.21)]. Default:  $5 \cdot 10^{-7}$  (in atomic units) .

See Eq. (3.21). Usually, this tag need not be modified from the default. Necessary for periodic boundary conditions only.

**Tag:** `hartree_partition_type` (`control.in`)

Usage: `hartree_partition_type` `type`

Purpose: Specifies which kind of partition function  $p_{\text{at}}(\mathbf{r})$  is used to split  $\delta n(\mathbf{r})$  into atom-centered pieces.

Restriction: Presently, `type` should have the same value as specified for integration using the `partition_type` keyword.

`type` : A string that specifies which kind of partition table is used. Default: `stratmann`

Usually, this tag need not be modified from the default. The same options are available as for the `partition_type` keyword (partition functions for three-dimensional integrands). See `partition_type` for details.

**Tag:** `hartree_radius_threshold` (`control.in`)

Usage: `hartree_radius_threshold` `threshold`

Purpose: Technical criterion to ensure the inclusion of atoms with a potentially finite real-space Hartree potential component in periodic boundary conditions.

`threshold` is a small positive real number. Default:  $10^{-10}$ .

Usually, this tag need not be modified from the default. Necessary for periodic boundary conditions only. For each atom, determines a safe real space outer radius based on  $\text{erf}(r_{\text{outer}}/r_0) < \text{threshold}$ . This is then used to determine which atoms need be included in the second term (sum over atoms) of Eq. (3.19).

**Tag:** `legacy_monopole_extrapolation` (control.in)

Usage: `legacy_monopole_extrapolation` flag

Purpose: Specifies how the monopole ( $l = 0$ ) part of the partitioned charge density is extrapolated to  $r = 0$  before transforming to a logarithmic grid to integrate the radial Hartree potential. If `.true.`, use the legacy variant, and an improved extrapolation otherwise.

flag is a Boolean. Default: `.false.`

The effect is generally very small, but for `light` grids, this can have some impact on total energies.

**Tag:** `l_hartree_far_distance` (control.in)

Usage: `l_hartree_far_distance` value

Purpose: Sets a maximum angular momentum beyond which the components of the analytic long-range Hartree potential will not be computed.

value is an integer number. Default: 10.

Usually, this tag need not be modified from the default. In Eq. (3.18), the multipole moments  $m_{at,lm}$  are determined by an explicit integration of the finite real-space density component  $\delta\tilde{n}_{at,lm}(r)$ . However, for very high  $l$ , even spuriously small density components ( $10^{-10}$  or lower) may be artificially weighted up in  $m_{at,lm}$ ; on a finite integration grid,  $m_{at,lm}$  becomes prone to numerical noise. Capping the evaluation of such high- $l$  components increases stability, but can be undone through `l_hartree_far_distance` if required.

**Tag:** `multip_moments_threshold` (control.in)

Usage: `multip_moments_threshold` threshold

Purpose: Implicitly defines the maximum angular momentum for which the analytical multipole components are non-zero at all.

threshold is a small positive real number. Default:  $10^{-10}$ .

Usually, this tag need not be modified from the default. Used only in the periodic case. If  $m_{at,lm}/r_{mp} < \text{threshold}$  for all  $l \geq l_{thr}$ , all analytical components beyond  $l_{thr}$  are considered zero in the real-space and Fourier parts of the long-range potential.  $r_{mp}$  is the radius determined by `multip_radius_threshold`.

**Tag:** `multip_moments_rad_threshold` (control.in)

Usage: `multip_moments_rad_threshold` threshold

Purpose: Defines the outer radius of the density components  $\delta\tilde{n}_{\text{at},lm}(r)$  for the purpose of determining the far-field moments  $m_{\text{at},lm}$ .

threshold is a small positive real number. Default:  $10^{-10}$ .

Usually, this tag need not be modified from the default. The outer radius is set where  $|\delta\tilde{n}_{\text{at},lm}(r)| < \text{threshold}$ . The actual  $m_{\text{at},lm}$  are then determined by inward integration from this point, using the standard relations of classical electrostatics.

**Tag:** `multip_radius_free_threshold` (control.in)

Usage: `multip_radius_free_threshold` threshold

Purpose: Technical criterion to define the outermost charge radius of the spherical free atom density  $n_{\text{at}}^{\text{free}}$

threshold is a small non-negative real number. Default: 0.0

Usually, this tag need not be modified from the default.

The free-atom radius inside the code is set to the radius where  $n_{\text{at}}^{\text{free}}(r)$  becomes smaller than threshold. Note that the actual extent of the free-atom charge can be influenced by the `cut_free_atom` keyword, and has ramifications not just for the electrostatic potential, but also for the initial charge density, and the partition functions for all integrals.

**Tag:** `multip_radius_threshold` (control.in)

Usage: `multip_radius_threshold` threshold

Purpose: Determines the (per-atom) radius outside of which the analytical multipoles  $m_{\text{at},lm}$  are used to construct the Hartree potential  $v_{\text{es}}(\mathbf{r})$

threshold is a small positive real number. Default:  $10^{-12}$ .

Usually, this tag need not be modified from the default. The outer radius is set where *all*  $\delta\tilde{n}_{\text{at},lm}(r) < \text{threshold}$  for a given atom. At a given integration point  $\mathbf{r}$ ,  $v_{\text{es}}(\mathbf{r})$  is assembled by evaluating Eq. (3.19). The second part (sum over atoms) is evaluated separately for each atom, and atoms outside the radius defined by `multip_radius_threshold`, the  $lm$  summation is performed using the analytical expression.

**Tag:** `multipole_threshold` (control.in)

Usage: `multipole_threshold` threshold

Purpose: Cluster case only – determines the distance beyond which an analytical component  $\delta\tilde{v}_{\text{at},lm}(r)$  of the real-space Hartree potential is considered zero.

threshold is a small positive real number. Default:  $10^{-10}$ .

Usually, this tag need not be modified from the default. Long-range Hartree potential components  $\delta\tilde{v}_{\text{at},lm}(r)$  are not evaluated for distances where  $\delta\tilde{v}_{\text{at},lm}(r) < \text{threshold}$ . This tag provides similar functionality as the `adaptive_hartree_radius_th` tag for

the periodic case (numerically different due to the absence of  $\text{erf}(r/r_0)$  in the cluster case).

**Tag:** `set_vacuum_level` (`geometry.in`)

Usage: `set_vacuum_level` *z*-coordinate

Purpose: Surface slab calculations only – defines a *z*-axis value that is deeply within the vacuum layer.

*z*-coordinate is a *z* coordinate value in the vacuum layer.

In the case of periodic surface slab calculations, this value defines the reference *z* coordinate that is used to define the work function (keyword `evaluate_work_function`) and/or the location of a dipole correction (electrostatic potential step) to offset a potential electrostatic dipole formed by a non-symmetric slab (keyword `use_dipole_correction`). As a requirement, the surface must be parallel to the *xy* plane. The chosen *z*-coordinate must be located deep in the vacuum, as far away as possible from any surface.

`set_vacuum_level` auto can be used instead to determine the vacuum level on its own.

*Omitting the keyword causes FHI-aims to automatically determine a suitable *z*.*

For any FHI-aims-version older than the git version 090211, this keyword has to be specified in the `control.in` file.

**Tag:** `use_dipole_correction` (`control.in`)

Usage: `use_dipole_correction`

Purpose: Surface slab calculations only – compensates a potential dipole field of non-symmetric slabs by an electrostatic potential step in the vacuum region.

Restriction: When specified for a charged periodic system, this keyword is currently disabled (see below).

If set, this option introduces an electrostatic potential step in the vacuum region of a surface slab calculation, to compensate for a potential surface dipole. The surface must be parallel to the *xy* plane (perpendicular to the *z* direction). The *z* location of the surface dipole must be provided by hand, by specifying the `set_vacuum_level` keyword.

In practice, the dipole correction calculates the gradient of only the long-range Hartree potential term of the Ewald sum (which is evaluated in reciprocal space). If the gradients on both sides of the vacuum level do not agree to better than 10 % (i.e., the potential is not linear in this range), the dipole correction is not computed, and a warning is issued instead.

*Attention:* This keyword is currently disabled for charged periodic systems. There is no formal physical reason to do so, but the Coulomb potential of a charged surface slab will reach far into the vacuum, most likely leading to a completely arbitrary dipole correction as a result. (The dipole correction will simply flatten out the potential wherever it is asked to do so, but for a charged surface, the residual Coulomb potential should not be flat.) In order to alert users of the problem, the code presently stops with a warning. If

you know what you are doing, the pertinent stop (one line) can always be commented out—if the code is recompiled, the method will be applied, even though the physical relevance of the result is uncertain.

---

**Tag:** `use_hartree_non_periodic_ewald` (`control.in`)

Usage: `use_hartree_non_periodic_ewald .true.`  
or: `use_hartree_non_periodic_ewald gridspacing value`  
or: `use_hartree_non_periodic_ewald .false.`

Purpose: This option is *experimental* and applies only to non-periodic calculations. In this case, the Hartree potential is decomposed according to Ewald's method.

This method accelerates the calculation of the Hartree term in case of large systems (more than 200 atoms) by using Ewald's decomposition combined with spatial interpolation, see section 3.7.1. The method can be switched on by using option `“.true.”`. In this case, a default grid spacing of 0.6 Å (= 60 pm) is used for the Cartesian grid. Other values for the grid spacing can be chosen with option `“gridspacing value”`. If this option is used, the method is switched on and the grid spacing is set to `value` in Å (= 100 pm). Finally, the method can be switched off with option `“.false.”`. However, since this is the default behaviour, it is not necessary to switch off the method explicitly.

## Subtags for *species* tag in `control.in`:

---

**species** sub-tag: `l_hartree` (`control.in`)

Usage: `l_hartree` value

Purpose: For a given species, specifies the angular momentum expansion of the atom-centered charge density multipole for the electrostatic potential.

value is an integer number which gives the highest angular momentum component used in the multipole expansion of  $\delta n(\mathbf{r})$  into  $\delta \tilde{n}_{\text{at},lm}(r)$  for the present species. *Must be specified.*

As pointed out in Ref. [12], our experience is that energy differences are usually well converged for  $l_{\text{hartree}}=4$ , and total energy convergence at the level of a few meV is reached at  $l_{\text{hartree}}=6$ . Only in exceptional cases should different settings be required.

## 3.8 Kinetic energy, scalar relativity, and spin-orbit coupling

For elements beyond approximately  $Z=30$ , relativistic effects near the nucleus cannot be neglected in an all-electron treatment—both for core, and for valence electrons. For the purposes of “everyday” matter, the full theory is given by Dirac’s four-component Equation, but in the “practice” of materials physics and chemistry, we still tend to think in terms of Schrödinger-like objects. A true four-component treatment is not implemented in FHI-aims, but the following standard levels of approximation are available:

- Non-relativistic kinetic energy (one or two collinear [spin](#) components of the Kohn-Sham orbitals)
- Scalar-relativistic kinetic energy expression (one or two collinear spin components).
- Perturbative spin-orbit coupling, a single correction step to the Kohn-Sham eigenvalues based on the Kohn-Sham orbitals from a non-relativistic or scalar-relativistic s.c.f. cycle. *Perturbative spin-orbit coupling in FHI-aims is a new feature, and should thus still be treated as an experimental implementation, with the appropriate caution when analyzing the results. Importantly, spin-orbit coupling is not yet available for periodic boundary conditions, and total energy gradients (forces) are also unavailable.*

### Scalar relativity

While the non-relativistic level of theory is exactly defined and will be the same in any first-principles implementation (at a complete basis set, all-electron level anyway), there are many different versions of scalar-relativistic approximations which can yield considerably different *total* energies for different systems. Their unifying feature is that any two scalar-relativistic methods should still yield the same energy *differences* for properties that concern valence electrons: Binding energies, valence eigenvalues, etc.

As outlined in Ref. [12], FHI-aims offers two reliable, effective one-component scalar relativistic treatments (by “effective one-component”, we mean that no coupling exists between the two spin channels of a calculation with collinear spin): “atomic ZORA” and “scaled ZORA”. (ZORA: “zero-order regular approximation”.) The potential energy surfaces and valence eigenvalues obtained from either one are sound, with total energy *differences* (e.g., between different structures) of scaled ZORA matching benchmark results obtained by other scalar-relativistic methods, perhaps a bit more closely in our tests.

To be precise, we have found it hard to arrive at a truly definitive “rating” of energy differences from atomic ZORA vs. scaled ZORA. The differences that we observe are usually well within the accuracy limits achievable with other scalar relativistic implementations [most importantly, (L)APW+lo as implemented in the Wien2k code]. In practice, any valence electron related results from atomic ZORA, scaled ZORA, or other scalar relativistic methods in the literature should be expected to agree with one another

up to benchmark accuracy level if all *other* computational tasks are numerically well converged.

What is clear, however, is that scaled ZORA *total* energies and *core* eigenvalues are, by construction, much closer to their counterparts from other scalar-relativistic methods than atomic ZORA. On the other hand, the computational overhead for scaled ZORA is significant, and calculating energy gradients (forces) is much more involved than for atomic ZORA.

As illustrated in more detail in Sec. 4.2, the correct handling of scalar relativity for arbitrary structures in FHI-aims is therefore twofold:

1. First, obtain the optimum geometry using the *atomic* zero-order regular approximation approximation (ZORA), which yields accurate conformational energy differences and geometries
2. Second, for the conformational energies and energy differences of the final structures, perform single-point calculations using the “scaled ZORA” [109] approximation.

Similarly, any other tasks involving forces should be performed using atomic ZORA, but remember that only energy *differences* between different structures are meaningful in this case.

## Perturbative spin-orbit coupling

In principle, spin-orbit coupling is a simple consequence of transforming Dirac’s Equation to a (two-component) Schrödinger like form. To achieve an effective one-component scalar relativistic form, the underlying two-component operator  $\boldsymbol{\sigma} \times \mathbf{p}$  ( $\boldsymbol{\sigma}$  is the vector of Pauli spin matrices and  $\mathbf{p}$  is the momentum operator) must be further replaced by  $\mathbf{p}$  alone. This is sufficient and, more importantly, efficient for most properties that concern valence electrons, but not, for example, for core levels: These will always exhibit a considerable splitting due to spin-orbit coupling.

A simple way to restore the effect of spin-orbit coupling  $\boldsymbol{\sigma} \times \mathbf{p}$  is to treat it perturbatively, i.e., as a single-shot postprocessing step after an otherwise converged scalar-relativistic self-consistent field cycle. This level of correction is essentially “complete enough” for any light and most heavy elements. However, we do point out that the eigenvalues of very heavy elements especially with  $p$  valence shells may require a spin-orbit treatment *beyond* perturbative, and some care is advised.

FHI-aims offers so-called quasi-degenerate (QD) perturbation theory to treat spin-orbit coupling. In a classical picture, an electron with velocity  $\mathbf{v}$ , moving in an electrical field  $\mathbf{E}$ , experiences a magnetic field  $\mathbf{B} \propto \mathbf{v} \times \mathbf{E}$ .

This leads to the interaction energy for the magnetic moment of the electron in that field:  $E = -\boldsymbol{\mu} \cdot \mathbf{B}$ . Using atomic units and  $\mathbf{E} = -\nabla v_{es}$  the interaction energy can be reformulated [76]:

$$h_{\text{SOC}}(\mathbf{r}, \mathbf{p}, \boldsymbol{\sigma}) = -\frac{1}{2c^2} \boldsymbol{\sigma} \cdot \mathbf{p} \times \nabla v_{es}. \quad (3.22)$$

In FHI-aims, the potential to generate the electrical field is currently chosen to be the mean-field (Hartree!) electrostatic potential  $v_{es}$ , the sum of electron and nuclear potentials. The exchange-correlation potential  $v_{xc}$  is not included. Note that this choice is not very important, as the truly meaningful potential gradients in any atomic structure are those due to the nuclear Coulomb potentials. In their immediate vicinity, all electronic potential components essentially act as relatively weak screening corrections.

*Quasi-degenerate perturbation theory* is based on Löwdin partitioning [71]. The spin-orbit interaction is seen as a perturbation to the scalar-relativistic Kohn-Sham-Hamiltonian  $\hat{h}^{sr}$ :

$$\hat{h} = \hat{h}^{sr} + \hat{h}^{soc}. \quad (3.23)$$

The scalar-relativistic set of eigenstates  $\{\psi^{sr}\}$  is then divided into several subsets  $\{\psi^{sr}\}_k$  in the following way. Two scalar-relativistic states  $i$  and  $j$  belong to the same subset if

1. their eigenvalues are close OR
2. their matrix element  $\langle \psi_i^{sr} | \hat{h}^{soc} | \psi_j^{sr} \rangle$  is comparable to the eigenvalue difference

which means that any two *different* subsets are weakly interacting in terms of  $h^{soc}$  and separated in energy. For each subset the Hamiltonian 3.23 is then “block-diagonalized” to a form  $\tilde{h}$  such that all matrix elements  $\langle \psi_{in\ subset} | \tilde{h} | \psi_{not\ in\ subset} \rangle$  vanish up to third order in  $h_{SOC}$ . For details see reference [113].

As a result, we obtain a perturbative correction to each scalar-relativistic eigenvalue, as well as a corrected set of Kohn-Sham orbitals. In particular, FHI-aims prints a zero-order corrected total energy by simply replacing the original (scalar-relativistic) sum of eigenvalues with its spin-orbit corrected counterpart.

## Tags for general section of `control.in`:

---

### Tag: `calculate_perturbative_soc` (`control.in`)

Usage: `calculate_perturbative_soc` `soc_tol` `ev_tol`

Purpose: *Experimental!* Calculate perturbative spin-orbit corrected eigenvalues.

`soc_tol` states in same subset if  $\left| \langle \psi_i^{sr} | \hat{h}^{soc} | \psi_j^{sr} \rangle \right| \times \text{soc\_tol} \geq |\epsilon_i - \epsilon_j|$   
`ev_tol` [in eV] states in same subset if  $|\epsilon_i - \epsilon_j| \leq \text{ev\_tol}$

*This feature is still experimental, only available for cluster calculations, and no total energy gradients (forces) are implemented as yet.*

Spin-orbit corrections can be added as a perturbative post-processing step after a non-relativistic, atomic ZORA or scaled ZORA scalar-relativistic calculation.

`calculate_perturbative_soc` requires the setting

`spin` collinear

to run.

A typical value for `soc_tol` is 3.0 .

A typical value for `ev_tol` is 0.2 eV .

*Since this is a new and experimental option, we recommend to monitor the effect of both tolerance settings carefully, for a system's eigenvalue spectrum as well as its total energy. We encourage feedback, especially if the above "typical values" prove to be insufficient for a specific example.*

In addition to the list of corrected eigenvalues and the new eigenvalue sum in the standard output, FHI-aims writes two separate output files with detailed information.

The file `SOC_result.out` contains the list of perturbed KS-eigenvalues including spin-up/down fraction of the corresponding state. The subsets used for the quasi-degenerate perturbation theory step and the perturbed eigenvectors can be found in `SOC_detail.out`. Errors are written to `SOC_warning.out`.

---

### Tag: `compute_kinetic` (`control.in`)

Usage: `compute_kinetic`

Purpose: *Experimental* - for test purposes, allows to compute the kinetic energy via the product of the kinetic energy matrix and the density matrix

This flag is presently kept for test purposes only (the electronic kinetic energy is separately computed and printed for each scf iteration anyway) but may be useful for some future modifications.

---

### Tag: `override_relativity` (`control.in`)

Usage: `override_relativity` flag

Purpose: If explicitly set, allows to override the stop enforced by the code when physically questionable `relativistic` settings are used.

`flag` is a logical expression, either `.true.` or `.false.` Default: `.false.`

For example, this will allow you to run a physically incorrect calculation of heavy elements (think Au) with Schrödinger's expression for the kinetic energy, instead of a scalar relativistic treatment. The results will be wrong, so this flag should only be set for test purposes. When set, the code assumes that the user must know what they are doing.

---

**Tag:** `relativistic` (`control.in`)

Usage: `relativistic` `r-type` `s-type` [`threshold`]

Purpose: Specifies the level of relativistic treatment in the calculation.

`r-type` is a string, specifying the basic approximation made.

`s-type` is a string, specifying whether a scalar treatment is desired (currently, only the scalar option is supported).

`threshold` is a small positive real number, allowing to reduce some integration effort.

Detailed expressions for the scalar relativistic treatments available here are given in Ref. [12]. We here only repeat the salient options and expressions. Possible options for `r-type` are:

- `none` : Non-relativistic kinetic energy. In this case, `s-type` and `threshold` need not be provided.
- `atomic_zora` : Atomic ZORA approximation as described in Ref. [12]. `threshold` need not be provided.
- `zora` The ZORA approximation is used throughout the s.c.f. cycle, followed by a "scaled ZORA" [109] post-processing step (rescaling of all eigenvalues). **WARNING:** Do not rely on intermediate, simple ZORA total energies, but only on the final, rescaled total energies instead!

Forces are only provided for `none` and `atomic_zora`.

**Remember to never take energy differences between calculations performed with different "relativistic" settings.**

In case of `zora`, the `threshold` option is required; it specifies the threshold value above which the difference between the sum-of-free-atoms ZORA expression and that for the actual potential during the s.c.f. cycle will be calculated. In areas of shallow potentials, where both expressions are substantially similar, this saves the extra integration effort associated with ZORA. For (very!) safe settings, `threshold` may be set to  $10^{-12}$ ; in our experience, also  $10^{-9}$  does not lead to any noticeable accuracy loss.

Default is `none` if all elements in the structure have  $Z < 20$  (no heavier than Ca). For all heavier elements, an explicit `relativistic` setting is required. For  $11 < Z < 20$ , the

setting `none` will be accepted, but a warning will be issued. For  $Z > 20$ , choosing `none` will cause the code to stop with an error message in order to avoid accidental calculations with incorrect relativity. If a non-relativistic calculation is still desired, for example for test purposes, this “stop” can be disabled by setting the flag `override_relativity`—but use this only if you know what you are doing.

## 3.9 Eigenvalue solver and (fractional) occupation numbers

With an updated Hamiltonian matrix  $h_{ij}$  and overlap matrix  $s_{ij}$  available at the end of a s.c.f. iteration ( $i, j$  run over all basis functions), or in the post-processing step of the calculation, FHI-aims updates the Kohn-Sham orbitals  $l$  (wave function coefficients  $c_{jl}$ ) by solving the following eigenvalue problem:

$$\sum_j h_{ij} c_{jl} = \epsilon_l \sum_j s_{ij} c_{jl} \quad . \quad (3.24)$$

In periodic boundary conditions, this eigenvalue problem is solved at every  $k$ -point, and  $k$  is implicitly included in the eigenstate index  $l$  above.

Since the basis size needed even for meV-converged accuracy in FHI-aims is rather small, and this size determines the dimension of  $h_{ij}$  and  $s_{ij}$ , the recommended eigenvalue solver(s) in FHI-aims are customized conventional solvers, employing the same basic algorithms as Lapack or the parallel ScaLapack implementation, but with significant scalability enhancements. Although these solvers scale strictly as  $O(N^3)$  with system size, their application becomes dominant only for systems above  $\approx 1000$  atoms (light elements) or  $\approx 500$  atoms (heavy elements, e.g. Au) in our experience.

The present section describes the available eigensolver options in FHI-aims, including the determination of a Fermi level and occupation numbers for all orbitals following the process.

FHI-aims also offers the possibility to solve a *constrained* eigenvalue problem, e.g., in order to restrict the number of spin-up or spin-down electrons in the basis functions of a given set of atoms. Since this functionality is experimental and for experienced users only, it is documented separately in Sec. 3.14.

Finally, we emphasize that the basis set in FHI-aims *is* non-orthogonal. For all practical production settings, this is not a problem, and in fact taken care of through the overlap matrix  $s_{ij}$  in Eq. (3.24) above. It is, however, still possible to generate an overcomplete, nearly ill-conditioned basis set in practical calculations, usually by specific, *deliberate* user action. The signature of such ill-conditioning are near-zero eigenvalues of  $s_{ij}$  (e.g.,  $10^{-5}$  and below). Possible reasons include: systematically constructed, deliberately overconverged basis sets for non-periodic calculations (not easy); excessively large cutoff radii in dense periodic structures together with very large basis sets (the density of non-zero basis functions per volume element increases as  $r_{\text{cut}}^3$ ); or, badly integrated, very extended basis functions (diffuse Gaussian basis functions without increasing `radial_multiplier` appropriately).

FHI-aims does include a number of safeguards against an ill-conditioned overlap matrix, most importantly the `basis_threshold` keyword that projects out the eigenvectors of the overlap matrix that correspond to its smallest eigenvalues, usually enabling a meaningful calculation anyway. However, to alert every user to the fact that their chosen basis set may be ill-conditioned, the code now stops when it encounters an overlap matrix with too low eigenvalues—unless the keyword `override_illconditioning` is deliberately set, indicating that the user knows what they are doing and wishes to

continue regardless.

---

## Tags for general section of `control.in`:

---

**Tag:** `basis_threshold` (`control.in`)

Usage: `basis_threshold` threshold

Purpose: Threshold to prevent any accidental ill-conditioning of the basis set.

threshold is a small positive threshold for the eigenvalues of the overlap matrix.

Default:  $10^{-5}$ .

Since NAO basis functions are situated at different atomic centers in a structure, they form a non-orthogonal basis set by construction. Usually, this is not a problem, since the non-orthogonality is naturally accounted for by inserting the overlap matrix  $s_{ij}$  into the Kohn-Sham eigenvalue problem, Eq. (3.24). For very large basis sets, this can lead to accidental ill-conditioning (some basis functions may be exactly expressible as linear combinations of some others).

This behavior is detected by directly inverting the overlap matrix, and computing its eigenvalues. If one or more eigenvalues are smaller than threshold, the corresponding eigenvectors are projected out of the basis before solving the Kohn-Sham eigenvalue problem, and the latter is solved after transforming to the reduced eigenbasis-set of the overlap matrix.

*Important change:* Even when `basis_threshold` is set, FHI-aims will automatically stop when a near-singular overlap matrix is detected. The user can still override this safeguard by setting the `override_illconditioning` keyword in `control.in` explicitly, but we do now do our best to alert the user to this condition.

---

**Tag:** `elpa_settings` (`control.in`)

Usage: `elpa_settings` setting

Purpose: Allows to determine the exact algorithm used in the ELPA eigensolver by hand.

setting is a descriptor (string) that selects certain aspects of ELPA. Default: `auto`

If the parallel ELPA eigensolver is used (see keyword `KS_method`), a number of choices are made automatically by default. The `elpa_settings` keyword allows to set some of these aspects by hand. Allowed choices for setting are:

- `auto` : The default. ELPA makes all its choices on the fly.
- `one_step_solver` : Only the one-step tridiagonalization (and corresponding back transformation) are used. This is usually the slower choice, but not always ...
- `two_step_solver` : Only the two-step tridiagonalization (and corresponding back transformation) are used. This is usually the faster choice, but not always ...

The `elpa_settings` keyword is particularly useful

- (i) if you already know what the faster choice is, and you wish to eliminate the extra test of the slower solver from your calculations, or
- (ii) if you suspect that one of the two solvers links to a buggy external(!) library. Lapack and BLAS implementations (still used in elpa) come from many vendors, they are often precompiled, and of course they *always* work—the computer vendor hopes so, after all. We have seen our share of bugs in external libraries (outside the control of FHI-aims), and sometimes, switching the algorithm to change the exact subroutines used can be a helpful backup check.

**Tag:** `empty_states` (`control.in`)

Usage: `empty_states` number

Purpose: Specifies how many Kohn-Sham states *beyond* the occupied levels are computed by the eigensolver.

number is the integer number of empty Kohn-Sham states *per atom* to be computed beyond the occupied levels.

For DFT-LDA/GGA, typically only a small (but non-zero) number of empty states is required to allow a complete determination of the Fermi level.

By default,  $(l+1)^2+2$  states are added *for each* atom in the structure, where  $l$  is the maximum valence angular momentum in the valence of that atom ( $l=0$  for hydrogen, but  $l=3$  for  $f$ -electron atoms and beyond).

For correlated methods including excited states (MP2, RPA,  $GW$ , ...), *all* available states should be included. To achieve this, set `empty_states` to a large number (safely larger than your basis set).

**Tag:** `fermi_acc` (`control.in`)

Usage: `fermi_acc` tolerance

Purpose: The precision with which the Fermi level for occupation numbers will be determined.

tolerance : Tolerance parameter for the zero point search of the equation  $\sum_i f_{\text{occ}}[\epsilon_F](\epsilon_i) - n_{\text{el}} = 0$ . Default:  $10^{-20}$ .

Usually, this tag need not be modified from the default. Within the (standard) Brent's method search for the Fermi level, tolerance has more than one function. Leave untouched unless problems arise.

**Tag:** `initial_ev_solutions` (`control.in`)

Usage: `initial_ev_solutions` number

Purpose: *Experimental!* Number of initial eigenvalue solutions using direct methods before switching on the lopcg-solver. Applies for both, Lapack and ScaLapack variants of the solver.

number is a positive integer. Default: 5.

**Tag:** `KS_method` (`control.in`)

Usage: `KS_method` `KS_type`

Purpose: Algorithm used to solve the generalized eigenvalue problem Eq. (3.24).

`KS_type` is a keyword (string) which specifies the update method used for the Kohn-Sham eigenvectors in each s.c.f. iteration. Default: `lapack_fast` or `scalapack_fast`, depending on available number of CPUs.

Available options for the eigensolver, `KS_type`, are:

- `lapack_fast` : **Preferred serial eigensolver implementation.** Lapack-based, and similar to the divide&conquer based standard solver provided by Lapack itself.
- `elpa` : **Preferred parallel eigensolver implementation. Uses the ELPA eigensolver library.** Same functionality as `scalapack_old`, however, substantially rewritten for an overall speedup and much improved scalability. Recommended on shared-memory platforms and distributed systems with any decent communication.  
See the `elpa_settings` keyword for some ELPA internals (usually determined automatically, but who knows).  
Note that you must set the shell variable `OMP_NUM_THREADS=1` prior to running FHI-aims on some platforms. (see Appendix A)
- `scalapack` : Synonymous with `elpa`.
- `scalapack_fast` : Synonymous with `elpa`.
- `lapack_old` : Expert solver provided by standard Lapack. This is stable and not a bottleneck in most standard situations.
- `lapack` : Synonymous with `lapack_fast`.
- `scalapack_old` : Fully memory-parallel implementation of the eigenvalue solver based on ScaLapack itself, scales much worse than our own `scalapack-fast`.
- `svd` : Effectively the same as `lapack_old`.
- `lopcg` : *Experimental – under development* Iterative, locally optimal preconditioned conjugate gradient eigensolver. Potentially useful for very large systems where `lapack` becomes a bottleneck. However, implementation without any serious testing—contact us if interested.

- `scalapack+lopcg` : *Experimental – under development*. Same as `lopcg`, but parallel with ScaLapack-type memory distribution.

The ScaLapack or ELPA eigensolvers are only available if ScaLapack support has been compiled into the FHI-aims binary—see the Makefile for more information.

In fact, the default parallel eigensolver in FHI-aims is the “ELPA” eigensolver, which uses some scalapack infrastructure but has been rewritten from the ground up for much improved parallel scalability.

On mildly parallel platforms (no more than tens of CPUs) and for normal system sizes (not significantly larger than  $\sim 100$  atoms), it is often acceptable to use the serial `lapack` eigensolver instead of the fully parallel `elpa`. Note that a (separate) parallelization over  $k$ -points will be performed in periodic systems in any case.

ELPA also allows calculations *without* explicitly collecting the resulting eigenvectors to each thread after the eigensolution is complete. This improves the memory efficiency especially in large-scale / massively parallel situations. For details, see keyword [collect\\_eigenvectors](#).

Prior to the solution of Eq. (3.24) using `lapack` or `elpa`, the overlap matrix  $s_{ij}$  is checked for ill-conditioning (see [basis\\_threshold](#) keyword). For very large basis sets or periodic calculations with many  $k$ -points, this criterion may trigger. In that case, the Hamiltonian matrix is transformed to the “safe” set of eigenvectors of  $s_{ij}$ , and the transformed eigenvalue problem is solved. If you suspect ill-conditioning to be a problem, it may also be helpful to increase the density of the 3D integration grids in order to minimize any numerical noise in  $s_{ij}$  and  $h_{ij}$ . That said: In our experience, ill-conditioning is not a problem with accurate basis sets in standard calculations; see Appendix A for some additional comments.

**Tag:** `lopcg_adaptive_tolerance` (`control.in`)

Usage: `lopcg_adaptive_tolerance` flag

Purpose: *Experimental!* Allows the `lopcg`-algorithm to dynamically adjust its convergence tolerance as  $\max\{0.01|\delta n|, \text{lopcg\_tolerance}\}$  where  $\delta n$  is the change in the electron density as recorded in the self-consistency cycles.

flag is a logical expression. Default: `.false`.

**Tag:** `lopcg_block_size` (`control.in`)

Usage: `lopcg_block_size` number

Purpose: *Experimental!* The maximal size of a block in `lopcg`-iteration.

number is a positive integer. Default: 1.

**Tag:** `lopcg_auto_blocksize` (`control.in`)

Usage: `lopcg_auto_blocksize` flag

Purpose: *Experimental!* Selects if the `lopcg` algorithm tries to find automatically a better blocksize than the maximal one by grouping close eigenvalues together.

`flag` is a logical expression. Default: `.false.`

---

**Tag:** `lopcg_preconditioner` (control.in)

Usage: `lopcg_preconditioner` type

Purpose: *Experimental!* For `KS_method` `lopcg`, specifies the preconditioner used.

type is a string, either `diagonal` (diagonal preconditioning matrix) or `ovlp_inverse` (use inverse of the overlap matrix for preconditioning).

---

**Tag:** `lopcg_start_tolerance` (control.in)

Usage: `lopcg_start_tolerance` tolerance

Purpose: *Experimental!* Sets the tolerance for starting the `lopcg`-solver using the change in the sum of eigenvalues as a criterion. The `lopcg`-solver is activated as set in `initial_ev_solutions` latest, but `lopcg_start_tolerance` may trigger it earlier.

tolerance is a double precision real. Default: 0.0

---

**Tag:** `lopcg_tolerance` (control.in)

Usage: `lopcg_tolerance` tolerance

Purpose: *Experimental!* Sets the convergence tolerance for the `lopcg`-solver.

tolerance is a double precision real. Default:  $10^{-6}$ .

---

**Tag:** `max_lopcg_iterations` (control.in)

Usage: `lopcg_tolerance` number

Purpose: *Experimental!* Sets the maximal number of iterations for one block in the the `lopcg`-solver.

number is an integer. Default: 100.

---

**Tag:** `max_zero` (control.in)

Usage: `max_zero` number

Purpose: Number of iterations allowed in Brent's method to find the Fermi level.

number is an integer number. Default: 200.

Usually, this tag need not be modified from the default. This limits the number of allowed iterations for the (standard) Brent's method search for the Fermi level. Leave untouched unless problems arise. Note that changing the values given for `occupation_type` or `empty_states` may be the true fixes if the search for a Fermi level really ever fails.

**Tag:** `occupation_acc` (control.in)

Usage: `occupation_acc` tolerance

Purpose: Accuracy with which the sum of calculated occupation numbers for a given Fermi level reproduces the actual number of electrons in the system.

tolerance is a small positive real number. Default:  $10^{-8}$ .

Usually, this tag need not be modified from the default. Determines the target accuracy for the Fermi level (calculated vs. actual number of electrons in the system) in the search using Brent's method. Note that changing the values given for `occupation_type` or `empty_states` may be the true fixes if the search for a Fermi level really ever fails.

**Tag:** `occupation_type` (control.in)

Usage: `occupation_type` type width [order]

Purpose: Determines the broadening scheme used to find the Fermi level and occupy the Kohn-Sham eigenstates.

type is a string which determines the desired broadening function. Default: gaussian

width specifies the width of the broadening function [in eV]. Default: 0.01 eV.

order is an integer, and only required to specify the order of type methfessel-paxton.

Based on the eigenvalues  $\epsilon_l$  of each s.c.f. iteration, the selected `occupation_type` determines the Fermi level  $\epsilon_F$  and occupies all Kohn-Sham states with fractional occupation numbers  $f_l(\epsilon_F)$  for the following electron density update. Detailed discussions can be found in Ref. [12] or other standard literature [61]. We only briefly list the available options for the occupation type here:

- gaussian : Gaussian broadening function [33]

$$f_l = 0.5 \cdot [1 - \operatorname{erf}(\frac{\epsilon_l - \epsilon_F}{\text{width}})]$$

- methfessel-paxton : Generalized Gaussian-type distribution functions of Methfessel and Paxton (see Ref. [77] for details). In practice, any order beyond 1 is not recommended.

- `fermi` : Formally correct finite-temperature broadening scheme [75]

$$f_i = \frac{1}{1 + \exp[(\epsilon_i - \epsilon_F)/\text{width}]}$$

However, to be useful in practice, `width` must take on values significantly greater than  $kT$  at room temperature, and therefore mostly loses its physical meaning. In practice, `fermi` broadening seems to lead to faster-increasing total energy inaccuracies than `gaussian` broadening, which is why the latter is preferred in FHI-aims.

*For metallic systems / systems with small HOMO-LUMO gap*, the availability of an occupation scheme with finite width (e.g., 0.1 eV) is critical to guarantee the stable convergence of the s.c.f. cycle. Especially for metallic systems, FHI-aims outputs an extrapolated total energy, which estimates the total energy for zero broadening based on the entropy of the electron gas [61, 35, 112]. This extrapolated total energy must only be used for metallic systems, not, e.g., for atoms with a decidedly discrete density of states.

*For non-metallic systems / systems with appreciable HOMO-LUMO gap*, the broadening width must be finite in order to guarantee the existence of a formal Fermi level, but not so large as to lead to any actual fractional occupation numbers. In our experience, the default width of 0.01 eV performs well for this purpose.

**Tag:** `override_illconditioning` (`control.in`)

Usage: `override_illconditioning` flag

Purpose: Allows to override a built-in stop and run with a nearly singular overlap matrix.

flag is a logical flag, either `.true.` or `.false.` Default: `.true.`

If the overlap matrix  $s_{ij}$  has an eigenvalue below `basis_threshold` or below  $10^{-5}$  (whichever is larger), FHI-aims will stop and warn the user of a potentially ill-conditioned basis set. Usually this situation can still be resolved by setting an appropriate value of `basis_threshold`, but anyone relying on this functionality should first check whether their “ill-conditioning” condition is not also due to another, inadvertent choice, such as an insufficient integration grid for very extended functions, or an excessively large cutoff radius in dense periodic systems (is it really necessary?).

In other words: By all means, override if you wish, but check first whether all computational settings are actually intentional and appropriate.

### 3.10 SCF Cycle: Initialization, density mixing, preconditioning, convergence

The preceding tasks (charge density update, Hartree potential, Hamiltonian and eigenvalue solver) are all methodologically simple, with well-defined standard choices, since they all relate to the densities and potentials within a single s.c.f. iteration of the Kohn-Sham equations only.

However, in order to run a complete, self-consistent Kohn-Sham or generalized ground state calculation, many such cycles must be performed. Beginning with well-defined initial criteria, self-consistency of the charge density and orbitals must be reached, and must be reached within a rather finite number of iterations.

For many standard problems in electronic structure theory—especially systems with a large, well-defined HOMO-LUMO or band gap—reaching self-consistency today presents essentially no problem, and is achieved to great accuracy already within  $\approx 10$  or so iterations.

However, in cases where the band structure is metallic, different charge or spin states are close to one another or in competition, there may be several self-consistent solutions, depending on the exact chosen initialization. Even worse, in such cases reaching even a *single* one of potentially several different self-consistent solutions can be problematic.

The present section summarizes all available options in FHI-aims to facilitate the self-consistent solution of any given problem in FHI-aims in as few iterations as possible, including:

- Initialization of the s.c.f. cycle
- Criteria for the convergence of the self-consistency solution
- Electron density mixing
- Electron density *preconditioning*

Please refer to Ref. [12] for a more exhaustive discussion of the physics / mathematics behind the individual choices laid out below.

*Important note:* The following settings are made or required by default.

- *The initial spin density must be specified in a spin-polarized calculation.* In spin-polarized systems, the choice of a good initial spin density can be critical for good convergence. For example, for a free atom, you might wish for a high-spin initial density according to Hund's rules. In a ferromagnetic Fe crystal, you might want to use a `default_initial_moment` of 2 (far lower than the Hund's rule value) to obtain fast convergence. In an antiferromagnetic Cr crystal, a ferromagnetic default initialization might do no good at all. And in a molecule with a single magnetic atom enclosed, you might want a spin-polarized initial moment only for that atom, but not for the surrounding molecule. In short, FHI-aims can not and should not guess the spin initialization for you. The program will now stop

if no initial moments of any kind are provided by the user. Setting either an overall `default_initial_moment` (in `control.in`), or at least one individual `initial_moment` tags in `geometry.in`, or both will allow you to run as usual.

- *Use of the Kerker preconditioner for periodic systems.* This option can greatly improve the s.c.f. convergence especially of large periodic systems (see `preconditioner` for more details). At the very least, it does not appear to do much harm, and is therefore now used by default in any periodic calculation. Should you encounter any difficulties, either turn the `preconditioner` off by hand, or play with associated screening momentum,  $q_0$  (default:  $q_0=2.0 \text{ bohr}^{-1}$ ).

Regarding options to converge the self-consistency cycle, note that one further important parameter is not covered here but instead in Sec. 3.9: The “broadening” of (fractional) occupation numbers around the Fermi level. Especially in metallic systems, this broadening must be large enough to prevent oscillations around the Fermi level, independent of the methods laid out below.

## Tags for `geometry.in`:

---

### Tag: `initial_charge` (`geometry.in`)

Usage: `initial_charge` charge

Purpose: Allows to place an initial charge on an `atom` in file `geometry.in`.  
charge is a real number. Default: 0.

The `initial_charge` keyword always applies to the last `atom` previously specified in input file `geometry.in`. The charge is introduced by using an ionic instead of neutral spherical free-atom density on that site in the initial superposition-of-free-atoms density. Note that initial charge densities are generated by the functional specified with `xc` for DFT-LDA/GGA, but refer to pw-lda densities for all other functionals (hybrid functionals, Hartree-Fock, ...).

---

### Tag: `initial_moment` (`geometry.in`)

Usage: `initial_moment` moment

Purpose: Allows to place an initial spin moment on an `atom` in file `geometry.in`.  
moment is a real number, referring to the electron difference  $N^\uparrow - N^\downarrow$  on that site. Default: Hund's rules, unless `default_initial_moment` is set explicitly.

The `initial_moment` keyword always applies to the last `atom` previously specified in input file `geometry.in`. The moment is introduced by using a spin-polarized instead of an unpolarized spherical free-atom density on that site in the initial superposition-of-free-atoms density. Note that initial charge densities are generated by the functional specified with `xc` for DFT-LDA/GGA, but refer to pw-lda densities for all other functionals (hybrid functionals, Hartree-Fock, ...).

Note that Hund's rules may be ambiguous. We therefore specify the default state for certain spin-polarized atoms explicitly (for initialization only!). For example, Cu is  $4s^13d^{10}$ , but could also be set (less efficiently) to be  $4s^23d^9$ .

## Tags for general section of `control.in`:

---

**Tag:** `charge_mix_param` (`control.in`)

Usage: `charge_mix_param` value

Purpose: Parameter for simple linear mixing of electron densities of previous and present s.c.f. iterations

value is a real number between 0. and 1. Default: Depends on chosen `mixer` algorithm.

See Ref. [12] for details regarding the available density mixing algorithms. In the simplest case of a linear `mixer`, value specifies a constant value  $\hat{G}^1$  to mix the output density of the Kohn-Sham Equations in iteration number  $\mu$ ,  $n_{\text{KS}}^{(\mu)}$ , with the (already mixed) *input* density that defined those equations,  $n^{(\mu-1)}$ :

$$n_{\text{dmp}}^{(\mu)} = n^{(\mu-1)} + \hat{G}^1 (n_{\text{KS}}^{(\mu)} - n^{(\mu-1)}). \quad (3.25)$$

If a `preconditioner` is specified, `charge_mix_param` defines an additional linear factor to that preconditioner. In case of a pulay `mixer`, all density residuals are mixed with this factor.

In general, the best choice for value is system-dependent, and also depends on the chosen `mixer` algorithm.

- In principle, a linear `mixer` will always converge with a sufficiently small value. In easy cases, we recommend `value`=0.1-0.2, but in difficult cases, “sufficiently small” can mean one to three orders of magnitude(!) lower, i.e., the process can be apallingly slow.
- For a straight pulay `mixer`, our default value is 0.6, which empiricallly works well for systems with a reasonable band gap / HOMO-LUMO gap; in metallic systems, e.g., `value`=0.2 is recommended.
- In metallic systems, density oscillations can occur from one iteration to the next (charge sloshing). This can be alleviated by a `preconditioner`. With a preconditioner and pulay `mixer` specified together, `value` is less critical and may be chosen well above 0.5 .

See also the `mixer` and `preconditioner` keywords. `charge_mix_param` sets a default for *all* possible mixing parameters in FHI-aims, but the following can be specified separately (additionally): `ini_linear_mix_param`, `ini_spin_mix_param`, `prec_mix_param`, and `spin_mix_param`. Note that it is not usually necessary to invoke those separate keywords unless the standard choices (`mixer` pulay, possibly with `preconditioner`) really fail.

---

**Tag:** `default_initial_moment` (`control.in`)

Usage: `default_initial_moment` moment

Purpose: For spin-polarized calculations, sets the default initial moment of the spin-polarized atoms that make up the initial electron density.

moment is either a string or a number that defines the desired initial number of electrons,  $N^\uparrow - N^\downarrow$ . Default: hund for isolated atoms. Zero otherwise.

Sets the default initial spin moment for all atoms whose `initial_moments` are not specified explicitly in `geometry.in`.

If there is at least a single `initial_moment` keyword specified in `geometry.in`, the `default_initial_moment` will be zero for all other atoms, for which no `initial_moment` is specified explicitly.

If no `initial_moment` is included in `geometry.in` at all, the `default_initial_moment` must be specified explicitly by the user for the code to run at all.

For most (bonded) systems, it is advisable to set the `default_initial_moment` to a numerical value close to what most atoms in the structure will do. For example, ferromagnetic Fe would be close to 2, whereas a large non-magnetic molecule would be better served with something close to zero.

For isolated free-atom calculations, `default_initial_moment` hund can be used. This will result in the usual high-spin atom initialization characteristic of free atoms.

Note that at least one moment in the system must be set to a non-zero value in order to reach any spin-polarized state at all. If the initial spin polarization is zero, the final s.c.f. result will also not be spin-polarized, no matter how magnetic the system is in reality.

**Tag:** `force_potential` (`control.in`)

Usage: `force_potential` type

Purpose: Determines how far / with which potential the Kohn-Sham equations are solved.

type is a string that determines the potential used. Default: sc

This option is not required under normal circumstances. It is mainly useful to produce / test a fast, non-selfconsistent solution for a superposition-of-atoms potential that yields only the sum of eigenvalues as a result. If a non-selfconsistent total energy is needed (correct only for the non-spinpolarized free-atom density!), running a normal calculation with `sc_iter_limit=0` is the better way.

Options for type are:

- sc: Self-consistent Kohn-Sham potential in each s.c.f. iteration
- superpos\_pot: Superposition of free-atom potentials, evaluation only once (no self-consistency cycle). *Restriction: This method works only for self-adapting angular\_grids (i.e., angular\_acc not equal zero for at least one species). This also means that the option will not perform well in periodic boundary conditions. A fix is simple, contact us if needed.*

- `superpos_rho`: Superposition potential created from sum of free-atom densities; evaluation only once (no self-consistency cycle)
  - `non-selfconsistent`: Same as `superpos_rho`.
- 

**Tag:** `ini_linear_mixing` (`control.in`)

Usage: `ini_linear_mixing` number

Purpose: If `mixer` is `pulay`, specifies simple linear mixing for a number of initial iterations.

number is the integer number of iterations for which linear mixing is done.  
Default: 0 .

Try only if the standard / preconditioned `pulay` `mixer` definitely fails. Keywords `ini_linear_mix_param`, `ini_spin_mix_param` can be used to specify separate mixing parameters for the initial linear mixing.

---

**Tag:** `ini_linear_mix_param` (`control.in`)

Usage: `ini_linear_mix_param` value

Purpose: Separate parameter for simple linear mixing of electron densities for `ini_linear_mixing`.

value is a real number between 0. and 1. Default: same as `charge_mix_param`.

`ini_linear_mixing` should only be tried if the standard algorithms provably fail. In that case, value should be relatively small.

---

**Tag:** `ini_spin_mix_param` (`control.in`)

Usage: `ini_spin_mix_param` value

Purpose: For spin-polarized calculations, separate parameter to mix the *spin* density during `ini_linear_mixing`.

value is a real number between 0. and 1. Default: same as `spin_mix_param`

`ini_spin_mix_param` should only be tried if the standard algorithms provably fail. In that case, value should be relatively small.

---

**Tag:** `mixer` (`control.in`)

Usage: `mixer` type

Purpose: Specifies the electron density mixing algorithm used to achieve fast and stable convergence towards the self-consistent solution.

type specifies the density mixing algorithm used and can be set to either `linear` or `pulay`. Default: `pulay`.

FHI-aims provides two mainstream density mixing algorithms across the s.c.f. cycle, type `linear` and `pulay`. We here only give a brief summary of options, please see Ref. [12] for further references and for the exact mathematical details.

For most practical purposes (non-pathological systems), Pulay's DIIS mixing algorithm [92] is robust and fast, and should be the algorithm of choice. For this algorithm, `n_max_pulay`  $n$  determines the number of past iterations  $\mu - k$  ( $k=1, \dots, n$ ) to be mixed with the Kohn-Sham output density of iteration  $\mu$ . `charge_mix_param` determines an additional (system-dependent) linear factor that is multiplied with the output density change of the Pulay mixer. Normally, this (and perhaps a `preconditioner`) is all you need to do to ensure convergence.

In some pathological cases, reaching self-consistency is a more tricky problem. Broadly speaking, these are systems with a small HOMO-LUMO gap (band gap) and/or several competing possibilities for a self-consistent solution. Specifically, these difficult cases include:

- Large metallic systems (e.g., slabs), where charge may “slosh” from one end of the system to another before reaching self-consistency. In that case, the `pulay mixer` may be used together with a large `charge_mix_param` and a `preconditioner` (see that keyword) to dampen the resulting oscillations. Also make sure that `occupation_type` is set to a sufficiently large broadening of occupation numbers near the Fermi level in metallic systems.
- Spin-polarized systems with competing spin states. A classic. If problems arise, playing with `ini_linear_mixing`, the `charge_mix_param` and `spin_mix_param` and further options listed in this section may help. Likewise, setting a specific `multiplicity` may be helpful. Finally, different `initial_moment` settings may easily switch between different metastable self-consistent spin states (e.g., ferromagnetic vs. antiferromagnetic), and should be tested separately if different competing spin states are suspected.
- Systems near a level crossing (even dimers, if two or more Kohn-Sham levels of different symmetry come close for a given binding distance). Apart from the usual mixing mechanisms, keyword `occupation_type` with a larger broadening near the Fermi level may help alleviate this situation.
- Spin-polarized free atoms. The simplest conceivable systems may exhibit unexpected problems towards self-consistency, likely because the electron density can rotate between several fully equivalent spin states. In particularly nasty cases, breaking the symmetry explicitly (e.g., using a specified `homogeneous_field` may be useful.

*In principle*, even in the toughest cases a `linear mixer` will always converge with a sufficiently small `charge_mix_param`. Unfortunately, “sufficiently small” can mean a `charge_mix_param` of  $10^{-2}$ - $10^{-4}$ , i.e., the process can be apallingly slow. Playing with the `pulay mixer` settings is usually the better strategy, unless a proof of principle is sought.

Note that a modification is needed when going beyond DFT-LDA/GGA (Hartree-Fock, hybrid functionals, ...). In that case, the density implicitly enters the two-electron exchange operator (via the density matrix,  $\hat{n}_{ij} = \sum_l f_l c_{il} c_{jl}$ , where  $i$  and  $j$  label basis functions, and  $l$  label the Kohn-Sham states), and should also be mixed.

By default, for `linear` mixing, we do not mix the exchange operator, unless keyword `use_density_matrix_hf` is enabled. The latter is the default if the `pulay` mixer is selected. Then, the density *matrix* is submitted to the same Pulay mixing factors as the normal charge density  $n(r)$  before constructing the exchange operator. Note that we do not have a formal density matrix available that corresponds to the *initial* superposition of free-atom densities, making this form of mixing slightly less efficient than for normal Kohn-Sham DFT-LDA/GGA.

---

**Tag:** `mixer_threshold` (`control.in`)

Usage: `mixer_threshold` keyword threshold

Purpose: Allows to cap the density step between two iterations rigorously by setting an explicit threshold.

`keyword` is a string, indicating whether the following is the charge or spin density threshold.

`threshold` is a real number, the maximum allowed change in the density norm (in electrons). Default: no thresholds.

This option is perhaps useful when there are definite convergence problems with the standard mixing algorithms, but can otherwise safely be ignored.

---

**Tag:** `n_max_pulay` (`control.in`)

Usage: `n_max_pulay` number

Purpose: The number of past iterations that the `pulay mixer` uses for density mixing.

`number` is the number of stored iterations used by the mixer. Default: 8

A larger number of stored iterations can sometimes lead to a stabilization of the mixing process. Choosing `number` too large (e.g., 20 and above), though, may destabilize the Pulay matrix, which can become near-singular.

Note that the storage effort associated with Pulay mixing is significant on systems with few CPUs / low memory. Each additional stored iteration requires the storage of two charge density residuals, and two times three charge density gradient residual components. For large systems, low memory, and overall stable mixing, reducing `n_max_pulay` may be a way to get a given calculation below the most difficult memory barriers.

---

**Tag:** `postprocess_anyway` (`control.in`)

Usage: `postprocess_anyway` boolean

Purpose: By default, FHI-aims simply stops if the SCF procedure does not converge. In particular, all desired postprocessing steps are skipped. If you do want postprocessing to be done anyway, set boolean to `.true.`.

boolean is either `.true.` or `.false.`. Default: `.false.`.

**Tag:** `prec_mix_param` (`control.in`)

Usage: `prec_mix_param` value

Purpose: Possible separate mixing parameter while the preconditioner is on.

value is a real number between 0. and 1. Default: same as `charge_mix_param`.

It's our tentative observation that a larger mixing parameter (0.5-0.8) is sometimes helpful with the `preconditioner`, but after the `preconditioner` is switched off, a smaller mixing parameter (as set by `charge_mix_param`) may be desirable. `prec_mix_param` can provide the needed separate setting, if desirable.

**Tag:** `preconditioner` (`control.in`)

Usage: `preconditioner` keyword [type] [value]

Purpose: "Master keyword" that precedes any information related to the preconditioner. May appear multiple times in `control.in`, in different contexts.

Restriction: Because it cannot simply be written in a density matrix formulation, the preconditioner has no effect on the density matrix entering the exchange operator for Hartree-Fock, hybrid functionals, etc.

keyword : A string, indicating the type of information following.

type : If required by keyword, a string with more details.

value : If required by keyword, a numerical value.

Default values:

- Non-periodic systems: `preconditioner` kerker off (no preconditioner).
- Periodic systems: `preconditioner` kerker 2.0 (Preconditioner with a momentum of  $q_0=2.0 \text{ bohr}^{-1}$ ).

See Ref. [12] regarding the mathematical definition of the Kerker-type [74, 81, 57] preconditioner, which is the currently implemented form.

See keyword `precondition_max_l` for the angular momentum cutoff specified for the kerker `preconditioner`.

keyword can have the following forms, controlling various aspects of the preconditioner:

- kerker :

- if followed by `off` : No preconditioner used.
- if followed by `value` : `value` is a real positive number, indicating the characteristic momentum  $q_0$  associated with the Thomas-Fermi type screening assumed in the preconditioner (in bohr<sup>-1</sup>). Typical values in the literature range around 1.0-2.0 bohr<sup>-1</sup>, but larger values may be useful in small clusters.
- `turnoff` : To avoid any residual influence on the s.c.f. cycle, the preconditioner can be switched off at a given level of s.c.f. convergence, leaving the remaining convergence to a pure pulay `mixer`. Possible types of turnoff criteria are:
  - `charge` : `value` refers to the root-mean-square deviation between  $n^{(\mu-1)}$  (the mixed and preconditioned *input* density to the Kohn-Sham Equations) and  $n_{\text{KS}}^{(\mu)}$  (the unmixed *output* density from the Kohn-Sham Equations). Default: `sc_accuracy_rho`.
  - `energy` : `value` refers to the total energy difference between two successive iterations [in eV].
  - `sum_ev` : `value` refers to the difference in the eigenvalue sums between two successive iterations [in eV].

All requested convergence criteria for the preconditioner must be fulfilled. If *no* explicit turnoff criterion is set, the `preconditioner` turnoff `charge`, `energy` and `sum_ev` defaults to the same values as `sc_accuracy_rho`, `sc_accuracy_etot`, and `sc_accuracy_eev`, respectively.

**Tag:** `precondition_max_l` (`control.in`)

Usage: `precondition_max_l` value

Purpose: Angular momentum cutoff used in the partitioned atom-centered real-space form of the kerker `preconditioner`.

`value` is an integer number, specifying an angular momentum. Default: 0.

We use a partitioned atom-centered multipole rewrite of the Kerker preconditioner in angular momentum space, similar in spirit to the Hartree potential (see Ref. [12] for details). In principle, `precondition_max_l` is thus the equivalent of the `l_hartree` angular momentum cutoff for the expansion. However, since we here precondition a density *difference* (which reduces to zero as we approach self-consistency), and since we are combatting charge sloshing across potentially faraway parts of the systems, preconditioning the atom-centered *monopole* component of the density difference (`value=0`) is often all that is needed for the preconditioner to work. This is also the numerically most efficient way of running the preconditioner.

**Tag:** `restart` (`control.in`)

Usage: `restart` file

Purpose: Saves and reads the final wave function of each scf-cycle to/from file.

file is a string, corresponding to the desired restart filename.

If file is not yet present, the calculation simply writes that file during the run. If file is already present, it is read and the wave function contained therein is used to restart the calculation, instead of a fresh superposition of free atoms initialization.

In parallel runs, there is one file for each process, numbered as fileXXX. See also `restart_read_only` and `restart_write_only`. There are limited checks on whether or not the restart file provided is actually from the same system, but ensuring that a given restart file works is mainly the user's responsibility. See also `restart_relaxations` and `MD_restart` for more information on the separate restarting process for relaxations and molecular dynamics respectively.

---

**Tag:** `restart_read_only` (control.in)

Usage: `restart_read_only` file

Purpose: reads the final wave function of the last scf-cycle in a preceding calculation from file.

file is a string, corresponding to the desired restart filename.

Similar to keyword `restart`, but does not overwrite file at any time (this may facilitate another restart from the same file later on).

---

**Tag:** `restart_write_only` (control.in)

Usage: `restart_write_only` file

Purpose: writes the final wave function of the last scf-cycle to file for a later restart.

file is a string, corresponding to the desired restart filename.

Similar to keyword `restart`, but does not read the restart file in case it already exists.

---

**Tag:** `restart_save_iterations` (control.in)

Usage: `restart_save_iterations` number

Purpose: writes restart information every number scf-iterations or at the end of each cycle.

number is the integer number of s.c.f. iterations after which the restart information is rewritten.

See also `restart`.

**Tag:** `sc_abandon_etot` (control.in)

Usage: `sc_abandon_etot` iter threshold

Purpose: If the s.c.f. cycle diverges, abort the s.c.f cycle after a specified number of iterations between which the total energy changed by more than a given threshold.

iter is an integer number of iterations after which the calculation is aborted.  
Default: 5 iterations.

threshold is a positive real number - if the total energy keeps changing by more than this number [in eV], the abort will be triggered. Default: 1000 eV.

This keyword allows to catch obviously ludicrous runs. If it triggers, something went seriously wrong during the mixing procedure and the settings for mixers, occupation broadening, preconditioner etc. should all be revisited very carefully.

The alternative setting `sc_abandon_etot` never switches the abort off, restoring the previous behaviour.

---

**Tag:** `sc_accuracy_eev` (control.in)

Usage: `sc_accuracy_eev` value

Purpose: Convergence criterion for the self-consistency cycle, based on the sum of eigenvalues.

value is a small positive real number [in eV], against which the difference of the eigenvalue sum between the present and previous s.c.f. iteration is checked.

Very sensitive criterion for s.c.f. convergence. Usually,  $\text{value}=10^{-3}$  eV is enough to indicate a reliable total-energy and force convergence. If value is set to zero or not given, the sum of eigenvalues will not be used as a convergence criterion.

---

**Tag:** `sc_accuracy_etot` (control.in)

Usage: `sc_accuracy_etot` value

Purpose: Convergence criterion for the self-consistency cycle, based on the total energy.

value is a small positive real number [in eV], against which the difference of the total energy between the present and previous s.c.f. iteration is checked.

The Harris-Foulkes form of the functional is used as the total energy in FHI-aims (see Ref. [12] for a brief discussion). A typical tight convergence criterion is  $\text{value}=10^{-6}$  eV. If value is set to zero or not given, the total energy will not be used as a convergence criterion.

---

**Tag:** `sc_accuracy_forces` (control.in)

Usage: `sc_accuracy_forces` value

Purpose: Convergence criterion for the self-consistency cycle, based on energy derivatives (“forces”).

value is a small positive real number [in eV/Å], against which the maximum difference of atomic forces between the present and previous s.c.f. iteration is checked.

*Attention:* If keyword `sc_accuracy_forces` is set in `control.in`, forces are by default computed, regardless of whether or not they are later needed. The rationale is that the only way to check a requested force convergence criterion is to compute the necessary forces, despite the added numerical effort. For single-point calculations (no relaxation required), `sc_accuracy_forces` should therefore not be set unless explicitly needed for some reason.

The default for `sc_accuracy_forces` depends on the required type of calculation:

- If no forces are needed / requested, `sc_accuracy_forces` is by default not checked.
- If forces are needed, the default for value is  $10^{-4}$  eV/Å. However, this is then still checked against the requested `relax_geometry` force accuracy below.
- If a geometry optimization is requested, the value for `sc_accuracy_forces` is checked against the requested convergence accuracy of keyword `relax_geometry`. If the requested accuracy for `relax_geometry` is less than 10 times the specified `sc_accuracy_forces` criterion, the latter value is set to 1/10 of the `relax_geometry` accuracy.

Calculating forces is relatively expensive, so this convergence criterion is only checked *after* the purely electronic / energetic criteria, `sc_accuracy_eev`, `sc_accuracy_etot`, `sc_accuracy_rho`, are all fulfilled. To avoid too many iterations with force computations before the forces are converged, it is important to set the other criteria (especially `sc_accuracy_eev`, which checks a non-variational quantity) to tight convergence, as indicated above. For `sc_accuracy_forces` itself, e.g., value= $10^{-4}$  eV/Å is a reliable and robust criterion to avoid noise in geometry relaxations.

**Tag:** `sc_accuracy_rho` (`control.in`)

Usage: `sc_accuracy_rho` value

Purpose: Convergence criterion for the self-consistency cycle, based on the charge density.

value is a small positive real number [in electrons], against which the volume-integrated root-mean square change of the charge density between the present and previous s.c.f. iteration is checked.

By default, FHI-aims checks separately the convergence of the charge density and the spin density, using the same criterion. Specifically, the *unmixed* output density of the Kohn-Sham Equations,  $n_{KS}^{(\mu)}$ , is checked against the input density  $n^{(\mu-1)}$  to the same

equations. A typical tight convergence criterion is `value=10-5`. If `value` is set to zero or not given, the charge density will not be used as a convergence criterion.

---

**Tag:** `sc_accuracy_stress` (`control.in`)

Usage: `sc_accuracy_stress` value

Purpose: Convergence criterion for analytical stress.

`value` is a small positive real number [in eV/Å<sup>3</sup>], against which the maximum difference of the analytical stress components between the present and previous s.c.f. iteration is checked. A negative number results in no convergence check.

*Attention:* If keyword `sc_accuracy_stress` is set in `control.in`, the analytical stress is by default computed, regardless of whether or not it is later needed. The rationale is that the only way to check a requested analytical stress convergence criterion is to compute the necessary analytical stress, despite the added numerical effort. For single-point calculations (no relaxation required), `sc_accuracy_stress` should therefore not be set unless explicitly needed for some reason.

The default for `value` is `not_checked`, i.e. the convergence of the analytical stress will not be checked. *However*, you have to ensure that other convergence criteria (especially `sc_accuracy_eev`, which checks a non-variational quantity) are set to tight values, as indicated above.

Calculating the analytical stress is relatively expensive, so this convergence criterion is only checked *after* the purely electronic / energetic criteria, `sc_accuracy_eev`, `sc_accuracy_etot`, `sc_accuracy_rho`, are all fulfilled. To avoid too many iterations with analytical stress computations before the analytical stress is converged, it is important to set the energetic criteria to tight convergence.

---

**Tag:** `sc_accuracy_potjump` (`control.in`)

Usage: `sc_accuracy_potjump` value

Purpose: Convergence criterion for the self-consistency cycle, based on the vacuum level potential shift.

`value` is a small positive real number [in eV], against which the difference of the dipole correction potential jump between the present and previous s.c.f. iteration is checked.

This keywords only makes sense (and is only accepted) for periodic slab calculations with the option `use_dipole_correction` set. If you are interested in the work function or vacuum level shifts explicitly, it is recommended to use this flag. A typical tight convergence criterion is `value=10-4`.

---

**Tag:** `sc_iter_limit` (`control.in`)

Usage: `sc_iter_limit` number

Purpose: Maximum number of s.c.f. cycles before a calculation is considered and abandoned.

number is an integer number. Default: 1000

`sc_iter_limit` is a keyword that should be set in every run. Note: **You must ensure for every run that the self-consistency cycle was actually converged.** If this is not the case, a loud warning is issued in the standard output of FHI-aims at the end of the s.c.f. cycle, and relaxations, molecular dynamics, and postprocessing may continue anyway depending on the `postprocess_anyway` setting.

If the end of the s.c.f. cycle is reached in this way, forces are computed regardless of whether the electronic convergence was reached.

**Tag:** `spin_mix_param` (`control.in`)

Usage: `spin_mix_param` value

Purpose: Separate parameter to mix the spin density between different s.c.f. iterations.

value is a real number between 0. and 1. Default: same as `charge_mix_param`.

`spin_mix_param` may be different from `charge_mix_param`, but there is not usually a clear recipe *how* it should be different. This option is thus only needed if the standard algorithms provably fail.

**Tag:** `switch_external_pert` (`control.in`)

Usage: `switch_external_pert` number type

Purpose: May be used as a combined parameter to switch on an artificial perturbing `homogeneous_field` only for a given number of iterations.

number is the integer number of s.c.f. iterations before the external perturbation is switched off.

type is a string. If set to `safe`, specific settings for the `occupation_type` and for the `homogeneous_field` are enforced (see below).

This is an experimental keyword that allows to switch on an initial `homogeneous_field`, e.g., to lock in the symmetry of a free atom in order to enforce smoother s.c.f. convergence. The field is switched off after `number` iterations, before self-consistency is reached.

If type is not `safe`, the actual value of `homogeneous_field` and `occupation_type` should be set explicitly in `geometry.in` and `control.in`, respectively. The default homogeneous field is  $10^{-3}$  eV/Å.

If type is set to `safe`, a homogeneous field of  $10^{-3}$  eV/Å and a Gaussian occupation with very small ( $10^{-5}$  eV) broadening are automatically enforced.

This flag is mainly used to artificially break the symmetry of spin-polarized free atoms

with open  $d$  and  $f$  shells, which are sometimes very hard to converge otherwise (see special cases listed for [mixer](#)). Remember that FHI-aims does not allow to enforce a given symmetry automatically.

---

**Tag:** `use_density_matrix_hf` (control.in)

Usage: `use_density_matrix_hf`

Purpose: Technical keyword that states that the density matrix is mixed prior to constructing the exchange matrix in hybrid functionals, Hartree-Fock, *et al.*

Default: When possible, `use_density_matrix_hf` is assumed.

## 3.11 Energy derivatives (forces, stress) and geometry optimization

With self-consistent Kohn-Sham orbitals, densities and total energies available, one of the primary tasks of electronic structure theory is to obtain energy *derivatives* with respect to the nuclear coordinates  $\mathbf{R}_{\text{at}}$ . First derivatives (“forces”) allow to find the optimum structure and molecular dynamics on the Born-Oppenheimer surface. Based on the structure optimum, second derivatives then enable the calculation of the (Born-Oppenheimer) zero-point vibrational energy of the nuclei, and of vibrational excitations (phonons).

The present section deals with the options available in FHI-aims for the computation of forces in FHI-aims, and with algorithms related to geometry optimization. Specifics regarding first-principles molecular dynamics are given in Sec. 3.12, and the computation of second energy derivatives (vibrational frequencies, zero-point energies, and oscillator strengths) by a finite-difference technique is covered in Sec. 4.6.

For most applications, simply specifying a convergence criterion `sc_accuracy_forces` will be enough to switch on the computation of forces itself. Reliable structure optimizations of atomic coordinates can then be obtained using the `relax_geometry` with the `bfgs` algorithm.

*Note that the default version of BFGS calls the trust radius enhanced variant of BFGS, also available as `trm`. A standard BFGS variant is also available as a fallback.*

In periodic calculations, you may also want to relax the unit cell shape itself. (However, be careful. In some cases, this can be a bad idea. For example, in high-symmetry structures a fit to Murnaghan’s Equation of state—see the utility provided in the `utilities` directory—will be faster and more accurate. Conversely, some low symmetry structures may have lattice instabilities and take you on an unwanted trajectory, such as from an fcc-ish lattice to a bcc-like one.) In any case, an additional keyword `relax_unit_cell` is needed there. It is chosen automatically whether to use the analytical or numerical stress tensor for the unit cell relaxation based on the computational settings. The numerical stress works for all settings (e.g. vdW corrections), but is very slow as the respective derivative is computed from finite differences. The analytical stress is faster, but uses more memory and is currently limited to LDA and GGA functionals without vdW corrections.

The remaining keywords documented below can be used to fine-tune the optimization, but will likely be not needed in most production calculations.

Note that analytic energy differences are not yet available for any `xc` method using the two-electron Coulomb-operator, i.e., Hartree-Fock, hybrid functionals, MP2 perturbation theory, RPA etc.

## Tags for geometry.in:

---

### Tag: `constrain_relaxation` (geometry.in)

Usage: `constrain_relaxation` constraint

Purpose: In `geometry.in`, fixes the position of the last specified `atom/lattice_vector` in a structure optimization.

constraint is a string, indicating what exactly will be constrained. Default: `.false.`

Allows to relax only parts of a structure, while keeping the rest at fixed positions. Currently, the following simple options for constraint are possible:

- `.true.:` All coordinates for this atom will be constrained.
- `.false.:` The relaxation of this atom will not be constrained.
- `x:` The  $x$  coordinate of this atom is not allowed to move.
- `y:` The  $y$  coordinate of this atom is not allowed to move.
- `z:` The  $z$  coordinate of this atom is not allowed to move.

*Attention:* If you wish to constrain more than one coordinate, the required constraints must be specified as separate lines, like this:

```
atom 0. 0. 0. Fe
  constrain_relaxation x
  constrain_relaxation y
```

In contrast, specifying two constraints in one line will *not* work. The second constraint would simply be ignored!

---

### Tag: `hessian_block` (geometry.in)

Usage: `hessian_block` i\_atom j\_atom block

Purpose: In `geometry.in`, allows to specify a Hessian matrix explicitly, with one line for each  $3 \times 3$  block. The option `block` consists of 9 numbers in column-first (Fortran) order. The  $3 \times 3$  block corresponding to `j_atom`, `i_atom` is initialized by the transposed of `block`. The Hessian matrix is input in units of  $\text{eV}/\text{\AA}^2$ .

If at least one `hessian_block` line is found in the file, the Hessian is constructed using this mechanism. So far there is no safe-guard from overriding Hessian blocks with subsequent lines with equal `i_atom`, `j_atom`.

There are two scripts in the `utilities` directory to automatically construct such Hessian matrix approximations. First, `conversions/thess2aims.py` converts a Tinker

generated Hessian matrix. Second, `Lindh.py` constructs a general purpose model matrix [70]. Please note that the `Lindh` model matrix is now also directly available with `init_hess Lindh`.

---

## Tags for general section of control.in:

---

### Tag: `bfgs_extrapolation_cap` (control.in)

Usage: `bfgs_extrapolation_cap` value

Purpose: If the `bfgs_textbook` algorithm of `relax_geometry` deviated from a line step of 1.0, sets a maximum for the line step during relaxation.

value is a real number  $\geq 1.0$  (dimensionless), the maximum allowed step when a variable line step used with BFGS. Default: 4.0

value is dimensionless, given in units specified by the BFGS Hessian matrix. value=1.0 corresponds to the default extrapolation of BFGS, assuming a perfectly quadratic energy surface.

---

### Tag: `bfgs_step` (control.in)

Usage: `bfgs_step` type [factor cap]

Purpose: Can specify a BFGS extrapolation with an adaptive line step instead of the default linestep for purely quadratic extrapolation.

type is a string, either `quadratic` or `mix`. Default: `quadratic`.

factor : If type is not `quadratic`, specifies a mixing factor to interpolate between the line step actually used in the last relaxation step, and the *a posteriori* estimated optimum line step. Default: 0.1

cap : If type is not `quadratic`, specifies an upper limit for the allowed line step.

---

### Tag: `clean_forces` (control.in)

Usage: `clean_forces` type

Purpose: Can remove small unitary force components (rotation and translation of the whole structure due to residual numerical noise) in relaxations.

type is a string, specifying whether and how any overall rotations / translations are removed.

The default for type depends on the exact circumstances (see below). The following choices exist:

- `none` : No removal of residual rotations and translations. This is the default if any external embedding fields or charges are specified in `geometry.in`.
- `sayvetz` : Non-periodic structures: Removal of rotations and translations by a formal projection [27, 100]. In *periodic* systems, only translations are removed.
- `fixed_plane` : *experimental* Simple alternative algorithm by constraining three atoms into a plane (implicitly constraining all others).

**Tag:** `compute_forces` (control.in)

Usage: `compute_forces` flag

Purpose: If `.true.`, switches on the computation of forces.

flag is a logical string, either `.true.` or `.false.`

Default: `.true.` if a geometry optimization or molecular dynamics run was requested, or if the `sc_accuracy_forces` convergence criterion was set. Otherwise, `.false.`

This flag allows to request an explicit force computation for an otherwise explicit single-point calculation. This is necessary for use with external tools that require forces, such as a finite-difference calculation of vibrational frequencies (see Sec. 4.6) or a transition state search (see Sec. 4.7). In these cases, keyword `final_forces_cleaned` should also be set.

---

**Tag:** `compute_numerical_stress` (control.in)

Usage: `compute_numerical_stress` flag

Purpose: If `.true.`, switches on the computation of the numerical stress tensor based on central finite differences.

flag is a logical string, either `.true.` or `.false.`

Default: `.true.` if a unit cell relaxation was requested and the computation of the analytical stress is not possible. Otherwise, `.false.`

If not further specified (by `delta_numerical_stress`) the default value for the scaling factor delta is set to  $10^{-3}$ .

---

**Tag:** `delta_numerical_stress` (control.in)

Usage: `delta_numerical_stress` value

Purpose: Specifies the scaling factor delta in the computation of the numerical stress tensor (`compute_numerical_stress`).

value is a dimensionless real number  $> 0$ . Default:  $10^{-3}$ .

---

**Tag:** `compute_analytical_stress` (control.in)

Usage: `compute_analytical_stress` flag

Purpose: If `.true.`, switches on the computation of the analytical stress tensor.

flag is a logical string, either `.true.` or `.false.`

Default: `.true.` if a unit cell relaxation was requested and computation is possible, or if the `sc_accuracy_stress` convergence criterion was set. Otherwise, `.false.`

This flag allows to request an explicit analytical stress tensor computation for an other-

wise explicit single-point calculation.

Currently, the calculation of the analytical stress is limited to LDA and GGA functionals without vdW corrections and is not possible with [load\\_balancing](#).

---

**Tag:** `calc_analytical_stress_symmetrized` (`control.in`)

Usage: `calc_analytical_stress_symmetrized` flag

Purpose: If `.false.`, calculates all 9 components of the analytical stress tensor. If `.true.` calculates only the upper triangle (6 components) of the tensor and copies the result to the lower triangle.

flag is a logical string, either `.true.` or `.false.` Default: `.true.`

Generally, it is sufficient to calculate the upper triangle of the tensor. This flag is mainly for debugging purposes.

---

**Tag:** `energy_tolerance` (`control.in`)

Usage: `energy_tolerance` tolerance

Purpose: Sets the energy amount by which a relaxation step can move upwards and is still accepted.

tolerance is a small positive real number, in eV. Default:  $2 \cdot 10^{-4}$  eV.

Small upward steps during relaxation may occur as a result of a slightly mis-guessed `bfgs_textbook` Hessian matrix somewhere along the path, or as a result of some residual numerical noise that leads to a discrepancy between energies and forces. In the present code version, such noise is always safely below the default `energy_tolerance` for reasonable settings. If exceeded, this may point to a different problem, simply increasing tolerance is not, in general, a good idea. For the `trm` optimizer, also see [harmonic\\_length\\_scale](#).

---

**Tag:** `final_forces_cleaned` (`control.in`)

Usage: `final_forces_cleaned` flag

Purpose: Decides whether spurious unitary transformations of the complete system (translations and rotations) are removed before the final output.

flag is a logical string, either `.true.` or `.false.` Default: `.true.`

This option affects directly the long-format (15 decimal) output of total energies and forces at the end of the `s.c.f.` cycle in the standard output file. If flag is `.true.`, the final output forces are “cleaned” using the sayvetz [27, 100] mechanism of keyword `clean_forces` (removal of translations and rotations for cluster geometries; only translations removed for periodic systems).

`final_forces_cleaned .true.` should be set for use with external tools that require forces, such as a finite-difference calculation of vibrational frequencies (see Sec. 4.6) or a transition state search (see Sec. 4.7).

**Tag:** `harmonic_length_scale` (`control.in`)

Usage: `harmonic_length_scale` length

Purpose: The trm/bfgs algorithm of `relax_geometry` judges a step by its energy gain. Usually, one simply uses the energy difference. For very short steps and rather light grids, however, it turns out that the quality of the energy is inferior to the quality of the forces. For steps shorter than `length`, do not look at the energy but use the harmonic approximation  $-\Delta\tilde{E} = (\mathbf{X}_2 - \mathbf{X}_1) \cdot (\mathbf{F}_2 + \mathbf{F}_1)/2$  as an estimate for the energy gain. If this procedure willfully accepts a step which increases the energy by more than `energy_tolerance`, the code stops to warn the user about the inconsistency between energy functional and forces.

`length` is a length scale in Å. Default: 0.025

Effectively, this flag switches from a real energy minimizer to a search for a stable zero of the force field for short step lengths.

**Tag:** `hessian_to_restart_geometry` (`control.in`)

Usage: `hessian_to_restart_geometry` flag

Purpose: Exports the current approximation to the Hessian matrix to `geometry.in.next_step` during a relaxation restart using `hessian_block`.

flag is a logical string, either `.true.` or `.false.` Default: `.false.`

**Tag:** `init_hess` (`control.in`)

Usage: `init_hess` diag [value] or: `init_hess` Lindh [add-value [thres]]

Purpose: Both the algorithms `bfgs_textbook` and `trm` in `relax_geometry` need an initial approximation for the Hessian matrix. With the option `diag`, value (in  $\text{eV}/\text{Å}^2$ , defaults to  $25 \text{ eV}/\text{Å}^2$ ) times the unity matrix is used. With `Lindh` the Lindh model matrix is used. For stability reasons, `add-value` (in  $\text{eV}/\text{Å}^2$ , defaults to  $0.005 \text{ eV}/\text{Å}^2$ ) times unity [70] is added. The parameter `thres` (default: 15.0) can be used to specify the accuracy of the Lindh matrix; only terms estimated to be larger than  $e^{-\text{thres}}$  are taken into account. Default is `init_hess Lindh 2`.

Alternatively, the initial Hessian can be specified explicitly by `hessian_block` in `geometry.in`.

**Tag:** `line_search_tol` (`control.in`)

Usage: `line_search_tol` tolerance

Purpose: During a `relax_geometry` `bfgs_textbook` geometry relaxation, rejects a relaxation step if the *a posteriori* estimate for the optimal relaxation step deviates by more than a given tolerance from the actually performed line step.

tolerance is a real positive relative tolerance (dimensionless in units of the BFGS Hessian). Default: 3.0

If keyword `relax_geometry` `bfgs_textbook` is used, `line_search_tol` will reject a relaxation step based on an *a posteriori* estimate of the true optimum step length (see Ref. [12] for details). In short, a step of length  $l$  in dimensionless units (units relative to the BFGS Hessian matrix) is rejected if

$$|l_{a\text{ posteriori}} - l_{\text{actual}}|/l_{\text{actual}} \geq \text{tolerance}. \quad (3.26)$$

---

**Tag:** `line_step_reduce` (control.in)

Usage: `line_step_reduce` value

Purpose: Determines the behaviour of FHI-aims if a `relax_geometry` `bfgs_textbook` relaxation step was rejected.

value is either a string descriptor, or a numerical value. Default: `automatic`.

During a structure optimization using the `relax_geometry` `bfgs_textbook` algorithm, FHI-aims checks if a performed relaxation step actually reduced the total energy compared to the previous step. The allowed upward tolerance is set by keyword `energy_tolerance`. If an upward step is encountered, FHI-aims first attempts to repeat the step with a smaller step length. If this smaller step also leads upwards, FHI-aims assumes that the BFGS Hessian has exceeded its validity limits, and reinitializes the Hessian before continuing.

If value is set to `automatic`, the BFGS step is repeated with an *a posteriori* estimated length  $l$  given by the previous (failed) step (see Ref. [12]).

Alternatively, value can be set to a real positive number  $<1$ , corresponding to the default factor by which the previous step length is reduced.

---

**Tag:** `max_atomic_move` (control.in)

Usage: `max_atomic_move` value

Purpose: Maximum allowed step length taken during relaxation.

value is a real positive upper bound for the maximum allowed change in single atomic coordinate, in Å. Default: 0.2 Å.

If the `bfgs_textbook`-predicted change in an atomic coordinate exceeds `max_atomic_move`, the length of the entire step (all coordinates) will be scaled down to not exceed the maximum allowed displacement.

---

**Tag:** `max_relaxation_steps` (control.in)

Usage: `max_relaxation_steps` number

Purpose: A structure optimization will be aborted after exceeding a prescribed maximum number of steps.

number is the prescribed maximum step number. Default: 1000 .

---

**Tag:** `min_line_step` (`control.in`)

Usage: `min_line_step` value

Purpose: Sets the minimum possible (dimensionless) BFGS linestep before the BFGS Hessian is reinitialized

value: A real positive number (between 0 and 1). Default: 0.1

See keyword `line_step_reduce`. If, after a rejected `bfgs_textbook` relaxation step, the reduced, dimensionless BFGS line step decreases below `min_line_step`, the BFGS Hessian matrix estimate is considered inappropriate for the present point in space, and the Hessian is reinitialized.

---

**Tag:** `numerical_stress_save_scf` (`control.in`)

Usage: `numerical_stress_save_scf` flag

Purpose: Controls if `constrain_relaxation` directives are used to determine implicitly if a component of the numerical stress has to be calculated. This greatly accelerates unit cells with high symmetries (e.g. orthorhombic).

flag is a logical string, either `.true.` or `.false.` Default: `.true.`

---

**Tag:** `orthonormalize_eigenvectors` (`control.in`)

Usage: `orthonormalize_eigenvectors` flag

Purpose: Specifies whether or not the wave function coefficients from the previous geometry will be re-orthonormalized before initializing a new relaxation step.

flag is a logical string, either `.true.` or `.false.` Default: `.true.`

The `orthonormalize_eigenvectors` keyword allows to reorthonormalize the converged self-consistent Kohn-Sham orbitals  $c_{jl}$  after a relaxation step. These are then used to reinitialize the electron density for the next relaxation step.

Due to the change in atomic positions, the wave function coefficients  $c_{jl}$  for the earlier geometry are no longer orthonormal after the relaxation step. The consequence is an initial electron density which no longer satisfies the correct electron count (i.e., the system may appear to be charged immediately after a relaxation step, although a neutral system was requested). In principle, this does not matter for the outcome of a calculation, since the self-consistent solution will be independent of the starting point. In many cases, there

is no clear benefit in terms of the s.c.f. convergence duration from orthonormalizing the  $c_{jl}$  prior to the reinitialization; however, some cases with unstable s.c.f. convergence may benefit significantly.

---

**Tag:** `relax_geometry` (`control.in`)

Usage: `relax_geometry` type tolerance

Purpose: Specifies if a structure optimization (geometry relaxation) is requested, and which.

type specifies the type of requested structure optimization. Default: none.

tolerance: Specifies the maximum residual force component per atom (in eV/Å) below which the geometry relaxation is considered converged.

Finds the nearest minimum of the Born-Oppenheimer potential energy surface for the nuclei.

For periodic calculations: If you are looking to relax not just atomic coordinates but also the unit cell shape (lattice vectors), you do need to specify an additional keyword: `relax_unit_cell`.

The presently supported options for type are none, `bfgs_textbook` and `trm` (which is now synonymous with `bfgs`).

- `bfgs` or `trm` uses a trust radius method enhanced version of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm (see Ref. [83], which was the basis for Jürgen Wieferink's code effort in this area). In our tests, this version appears to give the fastest convergence reliably.
- `bfgs_textbook` invokes a Cartesian BFGS optimization algorithm that adheres strictly to the textbook [91]; we note that our coded version is based on the concise description found at Wikipedia. See Ref. [12] for some additional details of our version.

The relaxation algorithm can be greatly sped up by using a somewhat intelligent guess for the Hessian matrix used in the initial step. By default, FHI-aims now sets the general purpose model matrix due to Lindh and coworkers [70] with a slight modification. For more information see the `init_hess` keyword above.

A reliable force convergence criterion tolerance for most structures is  $10^{-2}$  eV/Å. For difficult cases (flat energy surface over a large distance, or when a reference geometry for accurate vibrational frequencies is required, see Sec. 4.6), a tolerance of  $5 \cdot 10^{-3}$  or even  $10^{-3}$  eV/Å may be required. However, note the warning below. Going much below these values may result in using a great deal of computer time for no physical benefit whatsoever—but for very small numerical noise (from the finite integration grids), the total energy will already be converged.

*Warning:* Be careful to *not* set the tolerance option too tightly by default in all your runs. The code may then spend a great deal of time groping around for a point with

extremely small forces, while the total energy improvement is safely less than a few hundredths of a meV even for large systems.

Of course, specifying very tight force convergence is easily possible for test purposes. One may then check the behaviour using a tool like the 'get\_relaxation\_info.pl' script to see how many relaxation steps are actually producing worthwhile output.

---

**Tag:** `relax_unit_cell` (`control.in`)

Usage: `relax_unit_cell` type

Purpose: Relaxes unit cell (lattice vectors) using the structure optimization as specified in `relax_geometry`.

type specifies the type of requested unit cell optimization. Presently supported options: `none`, `full`, `shape`. Default: `none`

Allows to optimize the lattice vectors of a periodic calculation, in addition to the normal atomic coordinates inside the unit cell. This keyword is not on by default, as automatically optimizing the unit cell of (say) a surface calculation could do a lot of unintended harm. Possible settings:

- `none` : Unit cell will be kept fixed, no optimization.
- `full` : All `lattice_vector` degrees of freedom will be relaxed, except those affected by explicit constraints.
- `shape` : *Experimental* All angles between lattice vectors will be constrained (kept fixed), only the lengths of each lattice vector are varied.

This keyword should be used only together with `relax_geometry`. Individual lattice vectors or its components can be constrained by using `constrain_relaxation`.

If the computation of the analytical stress is possible regarding the chosen computational settings, the analytical stress is used for the unit cell relaxation. Otherwise, the numerical stress is used. With `stress_for_relaxation`, one can explicitly choose either numerical or analytical stress for the unit cell relaxation.

If a unit cell relaxation produces strange results with the analytical stress, you might want to check the convergence of the analytical stress with `sc_accuracy_stress`.

---

**Tag:** `stress_for_relaxation` (`control.in`)

Usage: `stress_for_relaxation` type

Purpose: Use either numerical or analytical stress for unit cell relaxations.

type can be either `numerical` or `analytical`. Default: Chosen automatically based on computational settings.

To perform an actual unit cell relaxation, one has to set `relax_unit_cell`. If one chooses `analytical` but the computation of the analytical stress is not possible, FHI-aims will abort.

---

**Tag:** `restart_relaxations` (`control.in`)

Usage: `restart_relaxations` flag

Purpose: To save all necessary data to restart a bfgs relaxation without needing to recompute the relaxation path in DFT.

`flag` is a logical string, either `.true.` or `.false.` Default: `.false.`

This is specifically useful for long structure optimizations pieced together from several successive runs of FHI-aims. In order to restart from an already known estimated Hessian, rather from unity (the default in the absence of any previous knowledge), the Hessian can be rebuilt from a series of stored atomic coordinates and forces. Please note that this flag only works if the `geometry.in` file is replaced by the `geometry.in.next_step` file, which is updated during the structure optimization. See also `restart` for keeping the wave functions as well.

## 3.12 Molecular dynamics

FHI-aims provides the capability to run Born-Oppenheimer molecular dynamics. The necessary keywords are described in this section. A brief description of the physical algorithms implemented in FHI-aims can be found in Ref. [12]. For a truly thorough explanation of the underlying concepts, please refer to the standard literature, e.g., Refs. [31, 13].

### Wave function extrapolation

For molecular dynamics runs within the microcanonical ensemble (“*NVE*”) or for deterministic thermostats, the wave function can be extrapolated in order to reduce the computational effort to reach self-consistency. This section specifies *what* quantity is actually extrapolated and *how*.

Following the approach of Kühne *et al.* [63], we extrapolate the contra-covariant density matrix  $\mathbf{PS}$ , the product of the ordinary (purely contravariant) one-particle density matrix  $\mathbf{P}$  and the overlap matrix  $\mathbf{S}$ . The contra-covariant density matrix is then used to project new wave function coefficients from the last iteration  $\mathbf{c}_{\text{new}} = (\mathbf{PS})_{\text{extra}} \mathbf{c}_{\text{old}}$ . As a last step, the one-particle coefficients are orthonormalized.

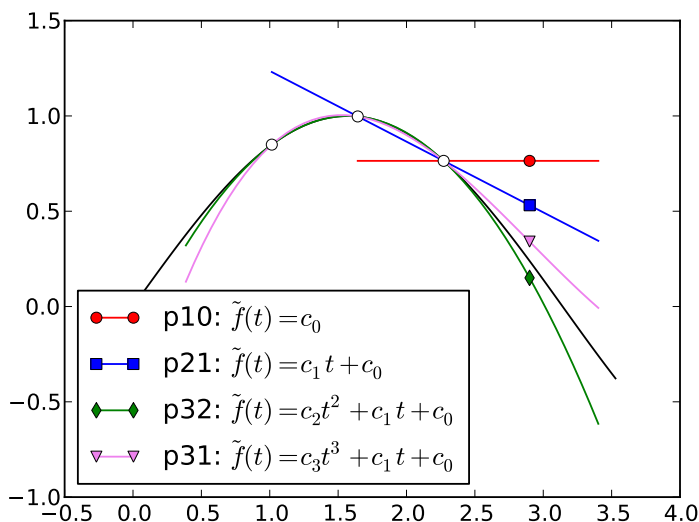


Figure 3.1: Different extrapolation schemes. The scheme “*pno*” refers to an  $n$  point scheme of order  $o$ , remaining orders used to fit odd components.

The extrapolation scheme we use is sketched in Fig. 3.1 for the example of an ordinary function  $f(t)$  in a real variable  $t$ . For a  $pno$  scheme (chosen with `wf_extrapolation polynomial n o`)  $n$  old iterations  $f(t_0 - k\Delta t)$ ,  $k = 1, \dots, n$  are stored. An ansatz function with  $n$  degrees of freedom is then used to fit all of these previous iterations and evaluated at  $t_0$  and the value used to initialize the SCF procedure.

The choice of the fitting function should aim at two targets: accurate extrapolation and

time-reversal symmetry. The parameter  $o$  specifies the order of the extrapolation. The higher  $o$ , the better the extrapolation gets with decreasing time stemp  $\Delta t$ . Time-reversal symmetry is useful to avoid energetic drifts for not-so-tight SCF accuracy settings. By adding odd terms (odd with respect to  $t_0 + t \leftrightarrow t_0 - t$ ), time-reversal symmetry is enhanced [60].

Therefore, in contrast to [93, 47], the  $(n - o - 1)$  remaining degrees of freedom are not resolved within a least-squares fit but instead to add fitting functions  $(t - t_0)^{2k+1}$  to enhance time-reversal symmetry and thus energy conversion. Please note, however, that time-reversal symmetry itself only enhances energy conservation and not necessarily dynamical properties of the trajectories.

If you are interested in energy conservation, “p31” is in general a good choice to start with. If a good initial guess for the SCF procedure is desired, you should give “p32” a try. Please note that deactivation of extrapolation (`wf_extrapolation none`) is actually the same as the “p10” extrapolation (see Fig. 3.1) and simply uses the one-particle coefficients of the last iteration are to initialize the SCF procedure.

## Tags for `geometry.in`:

---

### Tag: `velocity` (`geometry.in`)

Usage: `velocity vx vy vz`

Purpose: Specifies an initial velocity for the immediately preceding `atom` in file `geometry.in`.

`vx`, `vy`, `vz` :  $x$ ,  $y$ , and  $z$  components of the initial velocities, in Å/ps.

In `geometry.in`, the line containing the velocity must follow the line containing the `atom` that the velocity refers to. Note that, as in the case of relaxations, the molecular dynamics section of the FHI-aims output stream contains this information in the proper format.

## Tags for general section of `control.in`:

---

**Tag:** `MD_maxsteps` (`control.in`)

Usage: `MD_maxsteps` N

Purpose: Sets the maximal number of molecular dynamics steps.

N is an integer number. Default: -1 (infinite run).

A negative number signals that the ending criterion is not checked, in fact, the default setting is `N=-1`.

---

**Tag:** `check_MD_stop` (`control.in`)

Usage: `check_MD_stop` `.true.` / `.false.`

Purpose: if `.true.`, an MD calculation is stopped when a file `MD_stop` is generated

Default: `.true.`

---

**Tag:** `MD_MB_init` (`control.in`)

Usage: `MD_MB_init` Temperature

Purpose: Initializes random velocities in a molecular dynamics calculation using a Maxwell-Boltzmann distribution.

Temperature : Initial temperature in K. Default: No initial velocities.

This keyword is for a rough initialization only, and is overridden by any successful calls of the MD restarting procedure through `MD_restart`. The default initialization for all velocities is zero.

---

**Tag:** `MD_clean_rotations` (`control.in`)

Usage: `MD_clean_rotations` `.true.` / `.false.`

Purpose: if `.true.`, uses Sayvetz conditions to clean initial velocities from rotations

Default: `.true.` for non-periodic systems, `.false.` for periodic systems or if `relaxation_constraints` are used.

This option is useful for non-periodic systems, allowing to weed out some residual numerical noise in the forces. However, seeming rotations of the unit cell can easily appear in periodic systems for completely normal, physical reasons. Since this led to some confusion, this option is now actively disabled (code stops) for periodic systems. If you have a good reason to use this option in periodic systems, it can be re-introduced by hand.

---

**Tag:** MD\_thermostat\_units (control.in)

Usage: MD\_thermostat\_units units

Purpose: To allow user specification of the effective thermostat mass outside of the internal units.

unit : Unit of the effective mass of the Nosé-Hoover and Nosé-Poincaré thermostats – either  $\text{amu}\cdot\text{bohr}^2$  (mass-oriented specification) or  $\text{cm}^{-1}$  (frequency oriented specification, to allow to connect the thermostat mass to characteristic frequencies of the system). Default:  $\text{amu}\cdot\text{bohr}^2$  .

---

**Tag:** MD\_restart (control.in)

Usage: MD\_restart option

Purpose: controls the MD initialization from a molecular dynamics restart file.

Restriction: At present, this keyword does not produce proper results when switching ensembles between runs.

option is a string (see below). Default: .false.

Possible values for option are .true., .false., time\_restart, or filename. The default is .false..

The data for the molecular dynamics integrator is always written to a file aims\_MD\_restart.dat after each time step (regardless of whether or not keyword MD\_restart is used). If MD\_restart is set, this keyword controls the reading of such data from a previous run (if that exists), to either

- continue a previous run (.true.), or
- to use previous position / velocity information but reset all timings (time\_restart), or
- to begin a new run from scratch (.false.)

The default file name is used unless the user specifies a specific file filename from where the input is to be read.

**Important: If the starting structure and velocities are taken from the restart file, any exact positions noted in geometry.in are ignored.** However, the species identification, possible initial charges, and other per-atom information in geometry.in remain valid and must be present as always. If the option .true. is set, the molecular dynamics clock is also read from file, in all other restarts the clock is reset to zero.

---

**Tag:** MD\_restart\_binary (control.in)

Usage: [MD\\_restart\\_binary](#) option

Purpose: controls the format used for the molecular dynamics restart file.

option is a string (see below). Default: `.true.`

Possible values for option are `.true.` and `.false.`. The default is `.true.`. Use this flag to switch to the ASCII format in the MD restart file.

---

**Tag:** [MD\\_run](#) (control.in)

Usage: [MD\\_run](#) time ensemble [further specifications]

Purpose: Central controls of the physical parameters of a Born-Oppenheimer molecular dynamics run.

time : Requested MD time in ps.

ensemble : Ensemble specifications for [MD\\_run](#), listed as subkeywords to [MD\\_run](#) in a separate section below.

further specifications: See ensemble subkeywords below.

This keyword is the key control for all molecular dynamics. The runtimes time are specified in ps. There are five different possible ensembles, each of which require different options - as described in a separate subsection below.

When a *schedule* of different temperatures, thermostats etc is required within the same run (e.g., initialization followed by NVE, change of temperature, etc.), use the [MD\\_schedule](#) keyword instead of [MD\\_run](#).

---

**Tag:** [MD\\_schedule](#) (control.in)

Usage: [MD\\_schedule](#)

Purpose: Must be followed by specific segments of a Born-Oppenheimer molecular dynamics run.

Must be followed by an arbitrary number of lines [MD\\_segment](#), where different temperatures, thermostats, etc. may be specified for each segment of the run.

---

**Tag:** [MD\\_segment](#) (control.in)

Usage: [MD\\_segment](#) time ensemble [further specifications]

Purpose: Central controls of the physical parameters of a Born-Oppenheimer molecular dynamics run.

time : Requested MD time in ps.

ensemble : Ensemble specifications for [MD\\_run](#), listed as subkeywords to [MD\\_run](#) in a separate section below.

further specifications: See ensemble subkeywords below.

Keyword [MD\\_segment](#) must only appear in consecutive lines after an [MD\\_schedule](#) keyword. Instead of a single set of MD thermostats, temperatures etc. throughout the

simulation, this keyword allows to set specific values for only a segment of the full run.

---

**Tag:** `MD_time_step` (`control.in`)

Usage: `MD_time_step` `deltat`

Purpose: Set the time step for a molecular dynamics run, in ps.

Default: 0.001 (this is 1 fs)

---

**Tag:** `wf_extrapolation` (`control.in`)

Usage: `wf_extrapolation` `extrapolation_type`

Purpose: Used to specify the wave function extrapolation. Options are `polynomial n o` (an  $n$ -point polynomial extrapolation of order  $o$ , where the remaining degrees of freedom are used to enhance time reversibility), `niklasson06 n` (an  $n$ -point extrapolation as specified by Niklasson *et al.* [82]), `none` (same as `polynomial 1 0`), `linear` (`polynomial 2 1`), and `quadratic` (`polynomial 3 2`).

Default: `polynomial 3 1` for NVE, none otherwise.

The option `polynomial 3 1` strongly reduces a possible energy drift in NVE runs even for moderate force accuracy settings and additionally lowers the number of SCF cycles per time step.

*Warning:* This feature is experimental. It most probably will not enhance convergence of metallic systems and is not parallelized. The `niklasson06` schemes are not suitable for long runs of nontrivial physical systems because a regular reinitialization would be necessary.

---

**Tag:** `wf_func` (`control.in`)

Usage: `wf_func` `specifications`

Purpose: Used to activate the wave function extrapolation with fine grained control over the basis functions used for extrapolation. Each `wf_func` line adds one iteration to the extrapolation scheme and one fitting function. Options are `"constant"` (1), `"linear"` ( $t$ ), `"quadratic"` ( $t^2$ ), `"cubic"` ( $t^3$ ), `"polynomial n"` ( $t^n$ ), `"sin  $\omega$  unit"` ( $\sin \omega t$ ), and `"cos  $\omega$  unit"` ( $\cos \omega t$ ). The unit "unit" is either `"cm-1"` or `"fs"`. Additionally, `"none"` can be used to add one degree of freedom which is used to stabilize things in a least squares manner.

The same warning as for `wf_extrapolation` applies.

## Ensemble specification options for the MD\_run and MD\_segment keywords:

When used with a molecular dynamics schedule of time segments with different thermostats, the following subkeywords should appear behind an *MD\_segment* keyword instead of *MD\_run*.

---

### MD\_run sub-tag: NVE (control.in)

Usage: MD\_run time NVE

Purpose: Performs molecular dynamics in the microcanonical ensemble.

---

### MD\_run sub-tag: NVE\_4th\_order (control.in)

Usage: MD\_run time NVE\_4th\_order

Purpose: Performs molecular dynamics in the microcanonical ensemble, using a fourth-order integration method. The integrator used is called SI4 in Ref. [52]. This method is sometimes useful for longer time steps and for very accurate MD. Note that there are five force evaluations per time step, instead of the usual single calculation.

---

### MD\_run sub-tag: NVE\_damped (control.in)

Usage: MD\_run time NVE\_damped damping\_factor

Purpose: Performs microcanonical molecular dynamics, but dampens each velocity by a factor damping\_factor after each time step.

damping\_factor is the damping factor between different MD steps.

This option is a useful addition to the structural relaxation, which can be done in principle with a molecular dynamics run as well. In almost all cases, however, the BFGS algorithm as called with the *relax\_geometry* keyword is preferable.

---

### MD\_run sub-tag: NVT\_andersen (control.in)

Usage: MD\_run time NVT\_andersen Temperature nu

Purpose: Run molecular dynamics with an Andersen stochastic thermostat.

Temperature is the simulation temperature in K.

nu : Probability that a given atom's velocity will be "reset" to a Maxwell-Boltzmann distributed value, per picosecond!

Andersen's [3] simple definition of a thermostat that randomly resets the velocity of individual atoms to a Maxwell-Boltzmann distributed value with a given frequency. This

is an example of a rather harsh thermostat.

---

**MD\_run sub-tag: NVT\_berendsen** (control.in)

Usage: MD\_run time NVT\_berendsen Temperature tau

Purpose: Molecular dynamics run using the Berendsen thermostat.

Temperature is the simulation temperature in K.

tau is a relaxation time of the thermostat, in ps.

Note that the Berendsen thermostat does NOT resemble any physical ensemble, but that it is a useful standard to initialize a molecular dynamics simulation. The relaxation time tau must be chosen by the user, there is no default. Remember that the special case  $\tau = \Delta t$  reproduces the correct temperature exactly at every time step and can be used for a very rough initialization.

---

**MD\_run sub-tag: NVT\_parrinello** (control.in)

Usage: MD\_run time NVT\_parrinello Temperature tau

Purpose: Molecular dynamics run using the Bussi-Donadio-Parrinello thermostat. [15]

Temperature is the simulation temperature in K.

tau is a relaxation time of the thermostat, in ps.

This is the Bussi-Donadio-Parrinello thermostat, as described in Ref. [15] It a) properly samples the canonical distribution and b) preserves time correlations. The relaxation time tau (in ps) must be chosen by the user, there is no default. The performance of the thermostat is claimed in Ref. [15] to be practically independent of the value of  $\tau$ , for condensed phases. For clusters, though, a proper tuning of tau might be necessary. When this ensemble is chosen, the conserved pseudo-Hamiltonian (as described in Ref. [15]) is printed in the output.

---

**MD\_run sub-tag: GLE\_thermostat** (control.in)

Usage: MD\_run time GLE\_thermostat Temperature Number\_of\_aux\_DOF

Purpose: Molecular dynamics run using the colored-noise thermostats based on the Generalized Langevin Equation, proposed by M. Ceriotti and coworkers [17, 19, 18].

Temperature is the simulation temperature in K.

Number\_of\_aux\_DOF (integer) is the number of auxiliary degrees of freedom for this thermostat, that specifies the dimensions of the input matrices

This is a flexible thermostat based on the Generalized Langevin Equation, that adds extra degrees of freedom to the equations of motion and has a frequency dependent

memory kernel, so that one can adjust the performance of the thermostat for different degrees of freedom in the system. It requires also matrices as inputs, for which we refer the reader to the keywords `MD_gle_A` and `MD_gle_C`. Please read references [17, 19, 18] and references therein before using this. It can be used to sample the canonical ensemble or to break detailed balance and simulate other conditions, including approximate quantum effects. It has a conserved quantity in the same spirit of the BDP thermostat.

---

**MD\_run sub-tag: NVT\_nose-hoover** (control.in)

Usage: `MD_run` time `NVT_nose-hoover` Temperature Q

Purpose: Run molecular dynamics with a Nosé-Hoover thermostat.

Temperature is the simulation temperature in K.

Q : Effective mass specification of the thermostat in units as specified by `MD_thermostat_units`.

Probably the most popular thermostat in the literature.

---

**MD\_run sub-tag: NVT\_nose-poincare** (control.in)

Usage: `MD_run` time `NVT_nose-poincare` Temperature Q

Purpose: Run molecular dynamics with a Nosé-Poincaré thermostat.

Temperature is the simulation temperature in K.

Q : Effective mass specification of the thermostat in units as specified by `MD_thermostat_units`.

*Due to numerical issues, this thermostat has been disabled for the time being.*

---

**Tag: MD\_gle\_A** (control.in)

Usage: `MD_gle_A` entries

Purpose: Required input for `GLE_thermostat`

entries contains all entries (`Number_of_aux_DOF` + 1) of one row of the matrix. One must repeat this flag for each row of the matrix, in order.

Units should be 1/ps, and the matrix can be downloaded from <http://gle4md.berlios.de/>. If you wish to model different dynamics (quantum effects, or thermalization of PIMD), one can also specify a C matrix with the keyword `MD_gle_C`.

---

**Tag: MD\_gle\_C** (control.in)

Usage: `MD_gle_C` entries

Purpose: Optional input for `GLE_thermostat`

`entries` contains all entries (`Number_of_aux_DOF + 1`) of one row of the matrix. One must repeat this flag for each row of the matrix, in order.

This matrix is optional for the usage of the GLE thermostats. If sampling the canonical ensemble it is not needed. Otherwise, it is. Units should be K, and the matrix can be downloaded from <http://gle4md.berlios.de/>.

---

**Tag:** `use_pimd_wrapper` (`control.in`)

Usage: `use_pimd_wrapper` hostaddress portnumber

Purpose: Interfaces FHI-aims to an internet socket based python wrapper code that does path integral molecular dynamics.

`hostaddress` accepts "localhost" or an IP address. `portnumber` should contain the number of the port that the wrapper is listening to.

Code is now fully compatible with the python PIMD wrapper developed by M. Ceriotti, J. More, and D. Manolopoulos. Use 'Makefile.wrappi' to compile the code with the necessary routines. Additional documentation on how to use FHI-aims as a driver for this wrapper can be downloaded with the wrapper itself.

### 3.13 Thermodynamic Integration

*Note added to the present manual: Albeit functional, the thermodynamic integration routines are still classified as “experimental”. Please contact carbogno@fhi-berlin.mpg.de, if you encounter any problems or if you have suggestions.*

FHI-aims provides the capability to compute the *anharmonic contributions* to the *Helmholtz free energy* of a system with the so called “thermodynamic integration” technique. For a truly thorough explanation of the underlying concepts, please refer to the standard literature, e.g., Refs [110, 36], since only a basic overview that sheds some light on the required input is given here.

#### Theory

In this brief introduction, we will focus on a perfect, periodic crystal, the *Helmholtz free energy*  $F(T, V)$  of which can be decomposed in three contributions:

$$F(T, V) = F^{el}(V) + F^{nu}(T, V) = F^{el}(V) + F^{qh}(T, V) + F^{ah}(T, V) . \quad (3.27)$$

$F^{el}(T, V)$  is the free energy of the electronic system, which can be assessed by *Mermin’s canonical generalization* of DFT [75]. For large band gap insulators, the electronic contribution is approximatively temperature independent and thus equal to the free energy of the electronic system at zero Kelvin (see [occupation\\_type](#)).  $F^{nu}(T, V)$  is the free energy associated to the nuclear motion on the Born-Oppenheimer energy surface  $V^{nu}(\vec{R})$ . In the limit of low temperatures, this contribution can be described within the quasi-harmonic model (see Sec. 4.6), i.e., by only accounting for small elongations  $\vec{U}$  from the equilibrium positions  $\vec{R}_0$  on the approximative harmonic potential

$$V^{qh}(\vec{R}) \approx V^{nu}(\vec{R}_0) + \frac{1}{2} \sum_{\substack{L,\alpha \\ N,M,\beta}} \left. \frac{\partial^2 V^{nu}(\vec{R})}{\partial (R_{0,L})_\alpha \partial (R_{N,M})_\beta} \right|_{\vec{R}=\vec{R}_0} (U_{0,L})_\alpha (U_{N,M})_\beta . \quad (3.28)$$

The free energy  $F^{qh}(T, V)$  associated to the motion on such a potential can be computed with the FHI-aims code, as discussed in Sec. 4.6. At large temperatures, however, the quasi-harmonic approximation is no longer justified, since the deviations from the equilibrium are not minute. In this case, the “thermodynamic integration” technique can be employed to compute the *anharmonic contributions* to the free energy

$$F^{ah}(T, V) = F^{nu}(T, V) - F^{qh}(T, V) . \quad (3.29)$$

For this purpose, the dynamics of the *hybrid* system that is characterized by the potential

$$V^\lambda(\vec{R}, \lambda) = \lambda V^{nu}(\vec{R}) + (1 - \lambda) V^{qh}(\vec{R}) \quad (3.30)$$

is inspected. The parameter  $\lambda$  appearing therein describes the linear interpolation between the full Born-Oppenheimer potential  $V^{nu}(\vec{R})$  and the quasi-harmonic potential  $V^{qh}(\vec{R})$ . The free energy  $F^\lambda(T, V, \lambda)$  associated to the motion on this *hybrid* potential is directly related to the *anharmonic contributions* via

$$F^{ah}(T, V) = \int_0^1 d\lambda \left( \frac{\partial F^\lambda(T, V, \lambda)}{\partial \lambda} \right) , \quad (3.31)$$

as the fundamental theorem of differential and integral calculus shows. The relation [110]

$$\frac{\partial F^\lambda(T, V, \lambda)}{\partial \lambda} = \left\langle \frac{\partial}{\partial \lambda} V^\lambda(\vec{R}, \lambda) \right\rangle_{V_\lambda} \quad (3.32)$$

allows to replace the integrand in Eq. (3.31) with a *canonical ensemble average*  $\langle \cdot \rangle_{V_\lambda}$ . If an ergodic thermostat is used (see Sec. 3.12), this ensemble average can then be substituted with a time average, so that the *anharmonic contributions* can be eventually expressed as [110]

$$F^{ah}(T, V) = \int_0^t dt' \frac{d\lambda}{dt'} \left( \frac{\partial}{\partial \lambda} V^\lambda(\vec{R}, \lambda) \right). \quad (3.33)$$

Within this approach it is thus possible to determine the *anharmonic contributions* to the free energy from an *ab initio* MD simulation, in which the parameter  $\lambda$  is adiabatically varied from zero to unity and/or vice versa.

## Tags for MD\_schedule section of control.in:

---

**Tag:** `thermodynamic_integration` (`control.in`)

Usage: `thermodynamic_integration`  $\lambda_{start}$   $\lambda_{end}$  QH\_filename V0

Purpose: Specifies the thermodynamic integration parameters for the immediately preceding `MD_segment` in file `control.in`.

$\lambda_{start}$ ,  $\lambda_{end}$  : Initial and final value for  $\lambda$  in the specific `MD_segment`.

QH\_filename : Name of the file containing the parametrization of the quasi-harmonic potential  $V^{qh}(\vec{R})$ .

V0 : Value of the Born-Oppenheimer potential  $V^{nu}(\vec{R}_0)$  in equilibrium  $\vec{R}_0$ .

In `control.in`, the line containing the parameters for the thermodynamic integration must follow the line containing the `MD_segment` that the thermodynamic integration refers to. Note that a `thermodynamic_integration` line must be provided for all segments (or none) within an `MD_schedule` section, as shown in the following example:

```
MD_schedule
# Equilibrate the system for 100 fs at 800 K with lambda = 0
MD_segment 0.1 NVT_parrinello 800 0.0010
  thermodynamic_integration 0.0 0.0 FC_file.dat -0.328E+07

# Perform the thermodynamic integration over 10 ps
MD_segment 10.0 NVT_parrinello 800 0.0010
  thermodynamic_integration 0.0 1.0 FC_file.dat -0.328E+07
```

**Tags for QH\_filename:**

Note that the file QH\_filename can be automatically generated with the methods discussed in Sec. 4.6. As a reference, the syntax of the file is given here in spite of that.

---

**Tag: lattice\_vector** (QH\_filename)

Usage: `lattice_vector` x y z latt\_index

Purpose: Specifies one lattice vector for periodic boundary conditions.

x, y, z are real numbers (in Å) which specify the direction and length of a unit cell vector.

latt\_index : Sequential integer number identifying the lattice vector.

Lattice vectors associated with the equilibrium geometry. Please note that this input has to be equal to the specifications in `geometry.in`.

---

**Tag: atom** (QH\_filename)

Usage: `atom` x y z species\_name atom\_index

Purpose: Specifies the equilibrium location and type of an atom.

x, y, z are real numbers (in Å) which specify the atomic position.

species\_name is a string descriptor which names the element on this atomic position; it must match with one of the species descriptions given in `control.in`.

atom\_index : Sequential integer number identifying the atom.

Equilibrium atom positions. Please note that this input has to be consistent with the specification in `geometry.in` (same number of atoms, same order, same species).

---

**Tag: force\_constants** (QH\_filename)

Usage: `force_constants` FC\_x FC\_y FC\_z atom\_j direction atom\_i

Purpose: Specifies the force constants, i.e., the change in the forces that occur if one atom is displaced in the unit cell.

FC\_x, FC\_y, FC\_z are the change in the forces in the respective cartesian coordinates.

atom\_j : is the index of the atom that is displaced.

direction : is the cartesian direction in which atom\_j is displaced.

atom\_i : is the index of the atom the forces refer to.

Equilibrium atom positions. Please note that this input has to be consistent with the rest of the specification in QH\_filename.

Example for a very basic file QH\_filename:

```

lattice_vector  3.987   3.987   0.000   1
lattice_vector  0.000   3.987   3.987   2
lattice_vector  3.987   0.000   3.987   3

atom            0.000   0.000   0.000   Al  1
atom            1.993   1.993   0.000   Al  2
atom            0.000   1.993   1.993   Al  3
atom            1.993   3.987   1.993   Al  4
atom            1.993   0.000   1.993   Al  5
atom            3.987   1.993   1.993   Al  6
atom            1.993   1.993   3.987   Al  7
atom            3.987   3.987   3.987   Al  8

```

```

# displace atom 1 -----
force_constants  5.739e+00  -7.069e-16  6.805e-16  1 1 1
force_constants -1.909e+00 -1.455e+00 -1.324e-16  1 1 2
force_constants  3.692e-01  2.618e-16  3.813e-16  1 1 3
force_constants -1.909e+00 -1.398e-16  1.201e+00  1 1 4
force_constants -1.909e+00  2.040e-16 -1.201e+00  1 1 5
force_constants  3.692e-01  1.423e-16 -7.205e-16  1 1 6
force_constants -1.909e+00  1.455e+00  9.268e-17  1 1 7
force_constants -3.934e-02  4.931e-18 -4.373e-18  1 1 8
force_constants -2.979e-17  5.004e+00  1.698e-15  1 2 1
force_constants -1.016e+00 -1.430e+00 -9.899e-17  1 2 2
.....
force_constants  1.675e-19 -3.460e-02 -1.160e-17  1 2 8
force_constants  3.498e-17  2.663e-16  5.373e+00  1 3 1
.....
force_constants -2.894e-16  3.914e-16  3.969e-01  1 3 8
# end atom 1 -----
# displace atom 2 -----
force_constants -1.909e+00 -1.455e+00  3.466e-17  2 1 1
.....
force_constants -8.873e-17  1.914e-16  3.969e-01  2 3 8
# end atom 2 -----
.....
# end atom 7 -----
# displace atom 8 -----
force_constants -3.934e-02  4.931e-18 -4.373e-18  8 1 1
.....
force_constants  3.498e-17  2.663e-16  5.373e+00  8 3 8
#-----

```

## 3.14 Electronic constraints

Most production calculations only require the converged ground state of a calculation, but in some cases, a deliberate deviation from the Born-Oppenheimer surface is desired. For example it may be desirable to fix the spin state of a spin-polarized calculation using a defined `multiplicity`.

More generally, intuitive chemical concepts may suggest the localization of a fixed given spin moment or number of electrons in one part of a system, and a different spin moment or number in another part. Examples include enforcing definite charges on ions in a system, or a desired spin moment on one particular atom.

The latter idea of partitioning different numbers of electrons or spin moments in *space* thought of as divided into different atoms is inherently ambiguous. Nonetheless, this is a classic intuitive picture of chemistry, and, at least in the limit of well-separated system parts, becomes exact.

For *non-periodic* geometries, FHI-aims implements the possibility of constrained calculations to enforce predefined electron numbers in different regions and/or spin channels. We follow the prescription of Behler et al. [10, 11], which assigns electrons to different atoms according to a Mulliken-like analysis, i.e., by partitioning the occupied Kohn-Sham eigenstates according to the occupation of different atom-centered basis functions.

For details regarding the method, we refer to the original references, but we add here a clear word of caution. The implemented partitioning is well-defined when the relevant parts of the system are far apart, and/or when relatively small, confined basis sets are employed. For large, overlapping basis sets, electrons may be *spatially* located at one atom even though they are *numerically* assigned to the basis functions of a different atom. In other words, the procedure becomes more ambiguous as the basis size and completeness increase. Contrary to the usual paradigm of electronic structure theory, it is not meaningful to converge constrained DFT calculations to the basis set limit *if* the individual pieces of the system are not very well separated.

To set up an electronic constraint, the only keywords normally required are `constraint_region` in `geometry.in` and `constraint_electrons` in `control.in`. The remaining keywords documented below are normally required only for experimental purposes or troubleshooting.

---

## Tags for geometry.in:

---

**Tag:** `constraint_region` (geometry.in)

Usage: `constraint_region` number

Purpose: Assigns the immediately preceding `atom` to the region labelled number.

number is the integer number of a spatial region, which must correspond to a region defined by keyword `constraint_electrons` in file `control.in`. Default: 1.

To divide up space into regions for the purpose of enforcing an electronic constraint, each atom in the structure is included in a `constraint_region`.

Simple example of an Na-Cl dimer (geometry.in):

```
atom 0. 0. 0. Na
    constraint_region 1
atom 0. 0. 3. Cl
    constraint_region 2
```

assigns Na to the first region and Cl to the second region of a constrained calculation.

The special case of only one region (e.g., for a fixed spin moment calculation) needs no explicit `constraint_region` labels. Note that, apart from an explicit setup by keyword `constraint_electrons`, the case of an *integer* fixed spin moment for the *whole* system (all atoms) can also be called by the shortcut `multiplicity`.

## Tags for general section of `control.in`:

---

**Tag:** `constraint_debug` (`control.in`)

Usage: `constraint_debug` flag

Purpose: If set, provides extra output that monitors the convergence of the local constraint potentials used to enforce the requested constraint.

flag is a logical string, either `.true.` or `.false.` Default: `.false.`

---

**Tag:** `constraint_electrons` (`control.in`)

Usage: `constraint_electrons` region n\_1 [n\_2]

Purpose: Fixes the number of electrons to n\_1 (in the spin-polarized case, to n\_1 in the spin-up channel and n\_2 in the spin-down channel) for a given region.

region is an integer number, corresponding to one of the regions listed as `constraint_region` in `geometry.in`.

n\_1 is the number of electrons in the corresponding region (the number of spin-up electrons in the case of a spin-polarized calculation).

n\_2 is the number of spin-down electrons in the corresponding region (only needed in the spin-polarized case).

This is the central keyword that can be used to define a strict constraint on the electron numbers associated (i) with the orbitals of a given subset of atoms ("region") and / or (ii) with a given spin channel. See the `multiplicity` keyword for a shortcut for fixed spin moment calculations with an integer spin multiplicity.

---

**Tag:** `constraint_it_lim` (`control.in`)

Usage: `constraint_it_lim` number

Purpose: For the determination of the constraint potentials in different `constraint_regions`, sets the maximum number of internal iterations before the search for a converged value is aborted.

number is an integer number. Default: 200.

The method to determine the constraint potentials that enforce the local electron / spin constraint is set by `constraint_mix`; for more than one active `constraint_region`, this determination is always iterative. Keyword `constraint_it_lim` sets the maximum number of iterations before this search is aborted in case of failed convergence (or too ambitious accuracy requirements from keyword `constraint_precision`).

---

**Tag:** `constraint_precision` (`control.in`)

Usage: `constraint_precision` tolerance

Purpose: Sets the precision with which each requested local constraint on the electron count must be fulfilled.

tolerance is a small positive real number. Default:  $10^{-6}$ .

---

**Tag:** `constraint_mix` (`control.in`)

Usage: `constraint_mix` factor1 [factor2]

Purpose: Mixing factors for the iteratively determined constraint potentials.

factor1 is a mixing factor for the iteratively determined constraint potentials (for spin-polarized calculations, the spin-up mixing factor).

factor2 is the mixing factor for spin-down constraint potentials in the case of spin polarized calculations.

Only meaningful for non-standard settings of `mixer_constraint`, irrelevant for the standard bfgs case.

---

**Tag:** `ini_linear_mixing_constraint` (`control.in`)

Usage: `ini_linear_mixing_constraint` number

Purpose: If keyword `mixer_constraint` is a Pulay mixer, initial linear mixing for a few iterations can be requested first.

number is the number of initial linear iterations.

Only meaningful for non-standard settings of `mixer_constraint`, irrelevant for the standard bfgs case.

---

**Tag:** `mixer_constraint` (`control.in`)

Usage: `mixer_constraint` type

Purpose: Sets the iterative algorithm to determine constraint potentials.

type is a string. Default: bfgs

This flag has nothing to do with electron density mixing or the electronic self-consistency loop. Instead, this defines the process to determine constraint potentials that enforce the requested electron number constraints, *if* the keyword `constraint_electrons` was invoked. This process happens in each s.c.f. iteration after the Hamiltonian matrix is known, in the course of solving the Kohn-Sham eigenvalue problem.

If more than one constraint regions are requested, determining the constraint potentials to enforce the local constraint on the electron numbers is an iterative process. Technically, FHI-aims supports three different algorithms for type :

- `bfgs` : The default. A BFGS algorithm that optimizes the constraint potentials to their nearest local optimum. Nothing else should be used unless for experimental

purposes.

- `linear` : Linear mixing algorithm to determine the constraint potentials.
- `pulay` : Pulay-type mixing algorithm to determine the constraint potentials.

Under normal circumstances, the `mixer_constraint` keyword should not be needed explicitly.

**Tag:** `n_max_pulay_constraint` (`control.in`)

Usage: `n_max_pulay_constraint` number

Purpose: If the pulay mixer is selected for `mixer_constraint`, sets the number of iterations to be mixed.

number is the number of mixed iterations. Default: 8.

Only meaningful for non-standard settings of `mixer_constraint`, irrelevant for the standard bfgs case.

**Tag:** `force_occupation_basis` (`control.in`)

Usage: `force_occupation_basis` `i_atom` `spin` `basis_type` `basis_n`  
`basis_l` `basis_m` `occ_number` `max_KS_state`

Purpose: Flag originally programmed to compute core-hole spectroscopy simulations. In practice, it constrains the occupation of a specific energy level of an specific atom, being also able to “break the symmetry” of an atom.

`i_atom` is the number of the atom on which the occupancy is constrained, as it is listed in the `geometry.in` file.

`spin` is the spin channel (e.g., 1 if only one spin channel).

`basis_type` is the type of basis which is used to force the occupation of the orbital (set it to `atomic`).

`basis_n` is the main quantum number for the state of interest.

`basis_l` is the orbital momentum quantum number for the state of interest.

`basis_m` is the projection of the orbital momentum onto the z-axis (-1, 0, or 1 for a p state).

`occ_number` is the occupation constraint for the chosen state.

`max_KS_state` is the highest energy Kohn-Sham state for forcing occupation.

Example:

```
force_occupation_basis 1 1 atomic 2 1 1 1.3333 6
```

## 3.15 Embedding in external fields

To simulate the effect of external field (for instance, to connect to a QM/MM embedding formalism), FHI-aims allows to add the effect of a homogeneous electrical field and/or point charges surrounding the molecule in question.

Note that these embedding charges are *in addition* to any `charge` specified in `control.in`, and *not* already included there. `charge` should equal only to the sum of charges of all nuclei in `geometry.in` minus the overall number of electrons in the system, but does not count any embedding charges specified by keyword `multipole`.

*This functionality is not yet available for periodic systems.*

## Tags for `geometry.in`:

---

### Tag: `homogeneous_field` (`geometry.in`)

Usage: `homogeneous_field E_x E_y E_z`

Purpose: Allows to perform a calculation for a system in a homogeneous electrical field  $\mathbf{E}$ .

$E_x$  is a real number, the  $x$  component of  $\mathbf{E}$  in  $\text{eV}/\text{\AA}$ .

$E_y$  is a real number, the  $y$  component of  $\mathbf{E}$  in  $\text{eV}/\text{\AA}$ .

$E_z$  is a real number, the  $z$  component of  $\mathbf{E}$  in  $\text{eV}/\text{\AA}$ .

---

### Tag: `multipole` (`geometry.in`)

Usage: `multipole x y z order charge`

Purpose: Places the center of an electrostatic multipole field at a specified location, to simulate an embedding potential.

$x$  :  $x$  coordinate of the multipole.

$y$  :  $y$  coordinate of the multipole.

$z$  :  $z$  coordinate of the multipole.

`order` : Integer number, specifies the order of the multipole (0 or 1  $\equiv$  monopole or dipole).

`charge` : Real number, specifies the charge associated with the multipole.

If the order of the multipole is greater than zero (presently, only monopoles or dipoles are supported), a dipole moment must be specified in addition to the data provided with the `multipole` tag itself. To that end, a line must immediately follow the original `multipole` line, adhering to the following format:

```
data m_x m_y m_z
```

Here,  $m_x$ ,  $m_y$ ,  $m_z$  are the  $x$ ,  $y$ , and  $z$  components of the dipole moment, in  $e\cdot\text{\AA}$ .

---

## Tags for general section of `control.in`:

---

**Tag:** `full_embedding` (`control.in`)

Usage: `full_embedding` flag

Purpose: Allows to switch between embedding of the full electronic structure (affecting the Kohn-Sham equations) or the embedding of an electronic density that is calculated without knowledge of the embedding potential.

`flag` is a logical string, `.true.` or `.false.` Default: `.true.`

For most purposes, embedding into an external potential will involve a change to the electronic structure of the structure which is embedded. However, in some instances one may wish to embed a given charge density *non-selfconsistently*, i.e., by calculating the electron density without an external field and then computing the energy of that unperturbed electron density in the external field.

This feature is useful if multipoles are located too close to the quantum-mechanical region of the calculation. These act as Coulomb-like potentials, just like any other potential in the systems. If there are basis functions that cover the location of that potential, some electronic charge may artificially become trapped there, creating a bad approximation to the core / valence electrons of an atom with a nucleus of charge `charge`. In most cases, this is clearly undesirable behaviour, and apart from that will create unwanted numerical noise since the electronic structure near the Coulomb-like singularity of the multipole will be represented solely by basis functions that are inadequate for this purpose in the first place.

---

**Tag:** `qmmm` (`control.in`)

Usage: `qmmm`

Purpose: Allows to compute Hellmann-Feynman like forces *from* the quantum-mechanical part of a structure *exerted on* the multipoles on an external embedding field.

Restriction: Works only for external monopole potentials.

For quantum-mechanics / molecular-mechanics (QM/MM) “hybrid” molecular dynamics simulations, one must evolve *both* the quantum and classical subsystems with time. In that case, it is necessary to know the derivatives of the quantum-mechanical total energy with respect to the positions of the *classical* multipoles (see keyword [multipole](#)), i.e., the forces on the multipoles that originate from the quantum-mechanical region.

The computation of these forces is switched on by adding the `qmmm` keyword to `control.in`. The actual QM/MM simulation must still be performed using an outside framework, for example ChemShell [102] that uses the energies and forces provided by FHI-aims as a “plugin”.

## 3.16 QM/MM Embedding

*For the 081912 release, consider this functionality absolutely experimental and contact the maintainers before using it.*

When simulating nanostructured surfaces, it may be favorable to avoid the standard supercell approach and rather make use of a QM/MM embedding approach. E.g. with clusters or molecules adsorbed, extensive supercells are required to avoid spurious interaction between the nanostructure and its periodic copies. This makes computations tedious.

In the QM/MM approach, one embeds a quantum mechanical (QM) region in an extended monopole field. To avoid spurious charge leakage out of the QM region positively charged monopoles in the vicinity are replaced by norm-conserving pseudopotentials [58]. Those pseudopotential files can be either generated manually with the open source program package FHI98PP [34] or downloaded from [http://www.abinit.org/downloads/psp-links/lda\\_fhi](http://www.abinit.org/downloads/psp-links/lda_fhi) or [http://www.abinit.org/downloads/psp-links/gga\\_fhi](http://www.abinit.org/downloads/psp-links/gga_fhi).

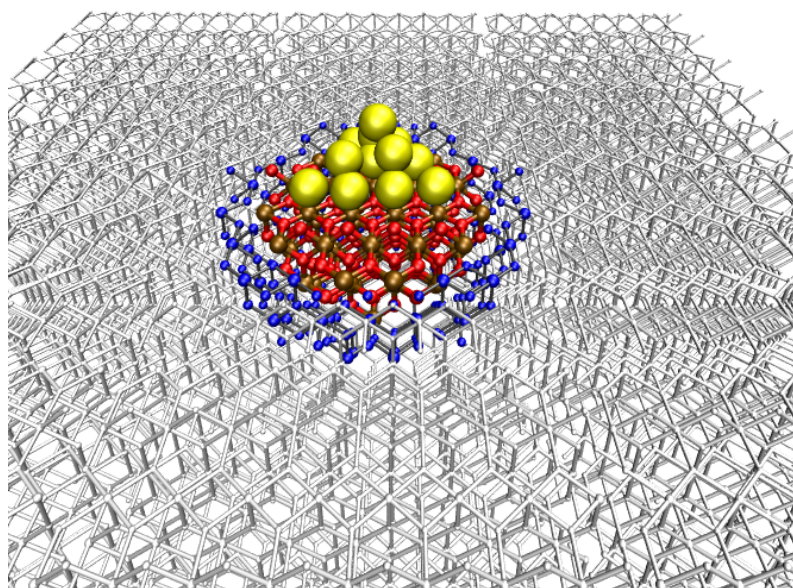


Figure 3.2: Example for QM/MM setup:  $Au_n@TiO_2$ . The adsorbed cluster and direct substrate vicinity defines the QM-region. The far field surrounding (grey particles) pictures a monopole field with formal charges (4+ for Ti and 2- for O). In the blue region, oxygen particles are still represented as monopoles, however Ti-cations are described with ionic pseudopotentials.

The QM/MM approach has the huge advantage ultimately also being capable to efficiently deal with **charged** systems, which will be a fundamental asset for the description of surface electrochemistry or photo-induced catalysis.

*QM/MM embedding is not applicable for metal substrates for physical reasons.*

## Tags for geometry.in:

---

### Tag: `pseudocore` (geometry.in)

Usage: `pseudocore` x y z species

Purpose: Places the center of a pseudopotential at a specified location.

x :  $x$  coordinate of the pseudocore.

y :  $y$  coordinate of the pseudocore.

z :  $z$  coordinate of the pseudocore.

species : string defining the name of the pseudoized species; this needs to correspond to name specified in control.in .

The keyword `pseudocore` should be used for those particles replaced by pseudopotentials, so cations. Anions are to be treated simply as monopoles, employing the `multipole` infrastructure.

## Tags for general section of control.in:

Only species data concerning the pseudoized species mentioned in `geometry.in` need to be appended in the `control.in`. Except for some mandatory changes (listed below) those species data are essentially the same as you can find them in the `species_default` folder. E.g. if you want to pseudoize for example titanium, take the `Ti` default file as a template. However, **the pseudoized species must not have any basis functions accept for the minimal basis**. The minimal basis is needed to construct the integration weights, however in order to exclude the minimal basis from the actual quantum chemical calculation, the flag `include_min_basis` needs to be set to `.false..`

Although nomenclature is misleading as it is chosen at the moment, you do NOT need the `qmmm` in order to make QM/MM embedding work.

Similar to all-electron `atom`, FHI-aims expects all atom specifications like `mass`, `nucleus`, information for the integration grid etc. Some additional flags need to be set that FHI-aims is able to realize them as pseudoized species.

---

### `species` sub-tag: `pseudo` (control.in)

Usage: `pseudo` string

Purpose: Parses the name of file the Kleinman-Bylander pseudopotential is written in

string name of file

FHI-aims expects the pseudopotential file to be in a specific formatting, namely the output format `*.cpi` of the generator program FHI98PP [34]. FHI98PP expects this file to be in the same folder as `control.in` and `geometry.in`.

---

**species sub-tag: pp\_charge** (control.in)Usage: `pp_charge` value

Purpose: Specifies the charge of the pseudoized ion.

value: any real value is allowed

`pp_charge` must be the charge which has been set in the generation of the pseudopotential and equals the number of pseudoized valence electrons. This parameter is needed for the far field extrapolation of the pseudopotential.

---

**species sub-tag: pp\_local\_component** (control.in)Usage: `pp_local_component` value

Purpose: Specifies which l-channel of the pseudopotential should act as the local component. Find a detailed theoretical background in [34].

value: integer value

The choice which l-channel should be the local component is essential for the performance of the pseudopotentials. Again, read [34] for further help.

---

**species sub-tag: nonlinear\_core** (control.in)Usage: `nonlinear_core` flagPurpose: when `.true.` FHI-aims expects and reads in a partial core density (and partial core density gradient) from the pseudopotential input file to take account of nonlocal core correction [72].flag is a logical expression, either `.true.` or `.false.` Default: `.false.`

```
[...]  
species          Ti_pseudo  
#   global species definitions  
nucleus          22  
mass             47.867  
  
pseudo          Ti.cpi  
pp_charge        4.  
pp_local_component  1  
nonlinear_core   .false.  
  
include_min_basis .false.  
  
[...]
```

Figure 3.3: Species data for a pseudoized titanium atom. Starting from the default species files only a few flags need to be added and the basis functions (accept the minimal basis) need to be removed.

### 3.17 $C_6/R^6$ corrections for long-range van der Waals (London dispersion) interactions

The correction improves the description of van der Waals (vdW) interactions in DFT. It is based on the leading-order  $C_6/R^6$  term for the interaction energy between two atoms. Both energy and analytic forces are implemented.

Two flavors of the correction are implemented:

- (1) The  $C_6$  coefficients and vdW radii are obtained directly from Hirshfeld partitioning of the DFT electron density. This scheme only requires a single damping parameter, which is fitted to binding energies of small organic molecules and hardwired in the code for PBE, PBE0, revPBE, AM05, BLYP and B3LYP functionals. For more information and citation see Ref. [106]. Both cluster and periodic cases are implemented.
- (2) The empirical  $C_6$  coefficients and vdW radii must be specified directly. This scheme is coded for maintaining compatibility with empirical  $C_6$  approaches. In actual applications, the usage of scheme (1) is advised, since it is significantly more accurate and less empirical.

## Tags for general section of control.in:

---

**Tag:** `vdw_correction_hirshfeld` (control.in)Usage: `vdw_correction_hirshfeld`

Purpose: Enables the vdW correction based on Hirshfeld partitioning of the electron density. If this keyword is set in a periodic calculation, the sum over atom pairs is done over successively larger supercells, until the energy is converged to the level set by `sc_accuracy_etot` and the forces (if requested) are converged to within `sc_accuracy_forces`.

No other input required.

---

**Tag:** `vdw_correction` (control.in)Usage: `vdw_correction`

Purpose: Enables the empirical  $C_6/R^6$  correction with the  $C_6$  coefficients and vdW radii specified by the user.

The user needs to specify the interaction parameters for *all* atomic pairs in the system (i.e. for CNOH, there are 10 atomic pairs). This is done by putting “`vdw_pairs N`”, where  $N$  is the number of pairs. This should be followed by  $N$  lines of “`vdw_coeff atomi atomj  $C_{6ij}$   $R_{ij}^0$   $d$ ””, where  $C_{6ij}$  is the  $C_6$  coefficient for the interaction between atomi and atomj,  $R_{ij}^0$  is the corresponding vdW radius and  $d$  is the damping function parameter. A choice  $d=20$  is suggested for all atomic pairs. An example for C-C interaction is: “vdw_coeff C C 30.00 5.59 20.0”.`

---

**Tag:** `vdw_pair_ignore` (control.in)Usage: `vdw_pair_ignore` species1 species2

Purpose: excludes the interaction between species1 and species2 from any  $C_6$ -correction, eg. such that metallic slabs are not affected internally by introducing  $C_6$ -interactions.

**Subtags for *species* tag in `control.in`:**

---

**species sub-tag:** `hirshfeld_param` (`control.in`)

Usage: `hirshfeld_param` C6 alpha R0

Default: the values outlined in Ref. [106]

Purpose: To explicitly allow setting the parameters for the Tkatchenko-Scheffler van der Waals correction.

## 3.18 Calculating nonlocal correlation energy within density functional approach

*Warning: This functionality is available in FHI-aims, but has not seen extensive testing by ourselves. It should therefore be treated as experimental. If you intend to use this functionality, by all means ensure that literature results obtained using this functional are reproducible using the implementation presented here.*

There are currently **two** separate working implementations of van der Waals DF in FHI-aims:

- Sec. 3.18.1: One version that allows post-processing only (i.e., compute the self-consistent density by another XC functional, then evaluate only the vdW-DF energy term again after the fact), using a Monte Carlo integration scheme. This version works for non-periodic as well as periodic systems. The original code was developed in the group of Claudia Ambrosch-Draxl at University of Leoben, Austria.
- Sec. 3.18.2: A second version that relies on an analytical integration scheme developed by Simiam Ghan, Andris Gulans and Ville Havu at Aalto University in Helsinki. This version allows non-self-consistent *and* self-consistent usage, as well as gradients (forces). It also has a number of numerical convergence parameters that can be adjusted.

### 3.18.1 Monte Carlo integration based vdW-DF

As a postprocessing step after a self-consistent calculation, the nonlocal part of the correlation energy can be calculated using the van der Waals density functional proposed by M. Dion et al. [24]. This task follows exactly the recipe presented in the original paper [24]. This calculation can be performed by choosing `ll_vdwdf` as a `total_energy_method` (please see Section 3.3).

**Important acknowledgment:** The Langreth-Lundqvist functional and basic Monte Carlo integration scheme used here was made available to us by the group of Claudia Ambrosch-Draxl and coworkers at University of Leoben, Austria. If you use this functionality successfully, please cite their work (currently, Ref. [80] and “unpublished”).

In order to perform the calculation, one should define an even spacing grid (followed by the `vdwdf` tag explained below), where the total charge densities of a system obtained after the scf cycle are projected.

In order to effectively solve the nonlocal correlation energy part, presented in Equation(1) of [24], the Monte Carlo integration scheme of Divonne integration method of CUBA library (Please visit <http://www.feynarts.de/cuba> for details).

This means that you should (yourself) compile the CUBA library as an external dependency to FHI-aims. The alternative Makefile `Makefile.cuba` contains examples of how to link FHI-aims to the CUBA library, and enable the vdW-LL functional

Parameters for tuning the performance of Monte Carlo integration are defined under

the `mc_int` tag explained below. Also, the kernel values are tabulated in terms of the parameters in the formula of Equation(14) of [24]. The aims package comes with the tabulated kernel data (a file called `kernel.dat`) and the name of the file should be included in `control.in`.

**Necessary input file:** In your FHI-aims distribution, a version of the `kernel.dat` file as well as an example `control.in` file should be located in subdirectory `src/ll_vdwdf` . For an actual FHI-aims program run, the `kernel.dat` file (currently, `kernel_my.dat` is provided) must be copied to your working directory and must be referenced in your `control.in` file, using the `kernel_data` sub-keyword of the `mc_int` keyword (see below).

## Tags for general section of control.in:

---

### Tag: mc\_int (control.in)

Usage: `mc_int` subkeyword(s)

Purpose: A line that begins with `vdwdf` is associated with the Monte Carlo integration performed to evaluate the non-local Langreth-Lundqvist functional. The `mc_int` keyword must be followed *on the same line* by a subkeyword that indicates the specific setting made here.

subkeyword(s) are one or more subkeywords or data for the Monte Carlo integration.

To use the Monte Carlo integrated Langreth-Lundqvist functional, more than one subkeyword for `mc_int` must be specified in the `control.in` file. See below for valid / necessary subkeywords. You may also want to check the documentation for the CUBA Monte Carlo integration library (<http://www.feynarts.de/cuba>) that you must have built and linked to the FHI-aims code in order to use the Langreth Lundqvist functional.

---

### Tag: vdwdf (control.in)

Usage: `vdwdf` subkeyword(s)

Purpose: A line that begins with `vdwdf` is associated with the non-local Langreth-Lundqvist functional. The `vdwdf` keyword must be followed *on the same line* by a subkeyword that indicates the specific setting made here.

subkeyword(s) are one or more subkeywords or data for the Langreth-Lundqvist functional.

To use the Langreth-Lundqvist functional, more than one subkeyword for `vdwdf` must be specified in the `control.in` file. See below for valid / necessary subkeywords.

## Subtags for vdwdf tag in control.in:

---

### vdwdf sub-tag: cell\_origin (control.in)

Usage: `cell_origin` *x y z*

*x y z* indicates the cartesian coordinates (in Å) of the origin of an even-spacing cubic cell. Default: 0.0 0.0 0.0.

This option is only valid for a cluster calculation. For the case of periodic system, a cell origin is automatically determined at the center of a supercell.

---

### vdwdf sub-tag: cell\_edge\_steps (control.in)

Usage: `cell_edge_steps`  $N_x$   $N_y$   $N_z$

Purpose: the total number of grids in each direction are defined by integer numbers,  $x$   $y$   $z$ .

---

**vdwdf sub-tag:** `cell_edge_units` (control.in)

Usage: `cell_edge_units`  $d_x$   $d_y$   $d_z$ .

Purpose: The real numbers  $d_x$   $d_y$   $d_z$  (in Å) define the length of grid units in each direction. Therefore, the full grid length is  $(N_x d_x, N_y d_y, N_z d_z)$ .

---

**vdwdf sub-tag:** `cell_size` (control.in)

Usage: `cell_size`  $L_x$   $L_y$   $L_z$

Purpose: this defines number of interacting cells in x, y, z directions for van der Waals interactions. This option is meaningful for periodic calculation.

$L_x$   $L_y$   $L_z$  are integer. Default: 0 0 0.

Note: As a temporary restriction, FHI-aims currently supports only grids with vectors aligning along  $x$ ,  $y$ , and  $z$  axes.

Calculations for periodic systems: In defining even spacing grid of a periodic system, only information of `cell_edge_steps` and `cell_size` (if needed) is necessary and other parameters will be automatically determined from that.

**Subtags for *mc\_int* tag in control.in:**

---

**mc\_int sub-tag: kernel\_data** (control.in)

Usage: `kernel_data` kernel.dat

Purpose: the name of the tabulated kernel file.

---

**mc\_int sub-tag: output\_flag** (control.in)

Usage: `output_flag` flag

Purpose: this controls output of Monte Carlo integration process, level 0 for no output, level 1 for "reasonable", and level 3 prints further the subregion results(if applicable). flag is an integer number. Default: 0

---

**mc\_int sub-tag: number\_of\_MC** (control.in)

Usage: `number_of_MC` N

Purpose: the total number of Monte-Carlo integration steps.

N is an integer number. Default: 5E5

---

**mc\_int sub-tag: relative\_accuracy** (control.in)

Usage: `relative_accuracy`  $E_{acc}$

Purpose: control the accuracy of Monte-Carlo integration performed by Cuba library.

$E_{acc}$  is a real number. Default: 1E-16

---

**mc\_int sub-tag: absolute\_accuracy** (control.in)

Usage: `absolute_accuracy`  $E_{abs}$

Purpose: control the error bar of nonlocal correlation energy.

$E_{abs}$  is a real number (in the unit of Hartree). Default: 1E-2

### 3.18.2 Analytic integration scheme for non-selfconsistent and self-consistent vdW-DF

This method calculates the non-local part of the correlation energy as described in [24] allowing for both non-self-consistent and self-consistent treatment. It works for both cluster and periodic geometries and can be used to compute forces. The implementation as well as the kernel function are from [39]. At each scf-cycle the following steps are performed:

- An octree is built to interpolate the current electron density to the new integration grid below.
- To each grid point of the main integration grid another grid of similar form is attached. The non-local correlation is then integrated on this grid using density and its gradient interpolated from the octree. In each node of the tree a tricubic interpolation is used.

The first step of building the octree is not parallel but the second step of integration is MPI-parallel the usual way.

## Tags for general section of control.in:

---

**Tag:** `nlcorr_nrad` (control.in)

Usage: `nlcorr_nrad` number

Purpose: Sets the number of radial shells used in the integration of the non-local correlation potential and energy. Default: 10

---

**Tag:** `nlcorr_i_leb` (control.in)

Usage: `nlcorr_i_leb` number

Purpose: Sets the index of angular Lebedev grid used in the integration of the non-local correlation potential and energy. Maximum value available is 15. Default: 7

---

**Tag:** `vdw_method` (control.in)

Usage: `vdw_method` type accuracy

Purpose: Sets the method for density interpolation for the integration of the non-local correlation potential and energy.

`type` is the method selected, either octree, mixed, or multipoles. Default: octree

`accuracy` applies to methods octree and mixed. It is the target accuracy of the interpolation. In case of multipoles all available multipoles are used. Default: 1E-6

The point of providing three different options for `type` is simply that any prospective user should test which one is fastest for a given problem. The difference is simply the style of integration of the non-local part. Ideally, the results should be the same. However, as always, please check in case of doubt.

### 3.19 Hartree-Fock, hybrid functionals, *GW*, *et al.*: All the details

The basic keywords to invoke different exchange-correlation methods (ground state and excited states) are given described in Sec. 3.3. Usually, invoking the relevant keyword together with the normal infrastructure required to run FHI-aims should be sufficient to produce a correct, converged result.

*For the periodic versions of Hartree-Fock and hybrid functionals, see also the dedicated next section, Sec. 3.20.*

*For relaxation with non-periodic functionals, we note that a specific version of the algorithms described below now works, but is still experimental. `RI_method LVL_fast` allows to use the relaxation infrastructure, but the accuracy of this RI version is slightly less than that of the normal RI-V method (see below). In addition, only small basis sets may safely be used for relaxation, as an untreated ill-conditioning issue for large basis sets affects the present version of the forces. In other words: Usable for some purposes, but be cautious.*

For methods that rely explicitly on a two-electron Coulomb operator (Hartree-Fock, hybrid functionals, *GW*, MP2, RPA, etc.) and/or a frequency-dependent response function (*GW*, RPA, ...), some considerable numerical trickery enters the computation in order to keep it efficient yet manageable for practical purposes. While we attempt to provide resilient, system-independent default settings, *as always in electronic structure theory we stress that a method should not be used as a blackbox method without an understanding of what is going on*, and without knowing how to test / repair things if something appears to go wrong.

The present section describes all numerical settings for the aforementioned exchange-correlation treatments. If you rely on any of these methods for production purposes: *Read your output file to understand what is going on, and use the below options if needed!*

Specifically, we describe:

- All settings that relate to the setup of the (over-)complete *auxiliary basis* that expands the products of pairs of basis functions into a separate basis to represent the Coulomb operator
- All settings that rely to the frequency grid and analytic continuation from the imaginary to the real axis in *GW* related methods.

Even if you do not know what this is all about, you *should* know that the “auxiliary basis” is determined as an overcomplete basis, and superfluous basis functions are then reduced out by a threshold criterion, using singular value decomposition. If this threshold is set too low, severe numerical instabilities may result. If, however, it is set to high, you may miss some of the Coulomb potential that you wish to describe. In case of doubt, it is a good idea to test the influence of this threshold parameter (within reason!) on a given calculation.

**Mathematical background:**

Any feature beyond standard DFT (e.g., HF, hybrid functional, MP2, *GW*, etc) requires the two-electron Coulomb repulsion integrals, and in FHI-aims an additional auxiliary basis set is introduced to deal with them. By utilizing the auxiliary basis functions the  $N_{\text{basis}}^4$  many 4-center integrals are reduced to  $N_{\text{basis}}^2 \cdot N_{\text{aux}}$  many 3-center integrals and  $N_{\text{aux}}^2$  many 2-center integrals ( where  $N_{\text{basis}}$  and  $N_{\text{aux}}$  are the numbers of regular basis functions and auxiliary basis functions, respectively). There are different ways to do so, and here we describe two versions of these, namely, the "V" and "SVS" [108] versions which have been implemented in this code. In the "V" version, the 4-center integrals are approximated by

$$(ij|i'j') \approx \sum_{\mu\nu} (ij|\mu) V_{\mu\nu}^{-1} (\nu|i'j'), \quad (3.34)$$

and in the "SVS" one,

$$(ij|i'j') \approx \sum_{\mu\nu} \sum_{\mu'\nu'} (ij\mu) S_{\mu\mu'}^{-1} V_{\mu'\nu'} S_{\nu'\nu}^{-1} (\nu|i'j'), \quad (3.35)$$

where  $i, j, i', j', \dots$  denote the regular basis functions and  $\mu, \nu, \dots$  denote the auxiliary basis functions. Here  $V_{\mu\nu}$  is the Coulomb repulsion integral between two auxiliary basis functions, and  $S_{\mu\nu}$  is the corresponding overlap integral.  $(ij\mu)$  and  $(ij|\mu)$  are the overlap and Coulomb repulsion between the regular basis orbital product  $\phi_i\phi_j$  and the auxiliary basis function  $P_\mu$  respectively. Eq. (3.34) and (3.35) are often referred to as resolution of identity (Refs. [14, 2, 108, 28] and others). In practice satisfactory accuracy can be gained with an auxiliary basis size  $N_{\text{aux}}$  of 4-5 times of  $N_{\text{basis}}$ .

How is the auxiliary basis functions constructed? In Fhi-aims these are built up as the "on-site" pair products of the regular basis functions (hence the auxiliary basis is also called as the product basis in this context). These products are then orthonormalized at each atom using the gram-Schmidt method. These auxiliary basis functions are hence atom-centered numeric functions with a radial function times spherical harmonics  $P_\mu(\mathbf{r}) = \frac{\xi(r)_{\text{at},n,l}}{r} Y_{lm}(\vartheta, \varphi)$ . The radial part of the auxiliary basis function is formally linked to that of the regular basis functions by

$$\{\xi_{\text{at},n,l}(r)\} = \{u_{\text{at},n_1,l_1}(r)u_{\text{at},n_2,l_2}, |l_1 - l_2| \leq l \leq |l_1 + l_2|\}. \quad (3.36)$$

In Eq. (3.36) we make it clear that the set of auxiliary basis functions centered on certain atom originates from the pair products of regular basis functions centered on the the same atom. The angular momentum of the auxiliary basis and those of the two constituent regular basis satisfy the triangular rule. The number of auxiliary basis function for a give  $l$  (enumerated by  $n$ ) is controlled by the allowed pairs of regular basis functions, and the accuracy threshold in the Gram-Schmidt orthonormalization. The parameters that controls the construction of the auxiliary basis functions can be found below.

For self-energy calculations a proper frequency grid is needed. For instance, in the *GW* approximation, the self-energy is given by

$$\Sigma(\mathbf{r}, \mathbf{r}'; i\omega) = \frac{i}{2\pi} \int d\omega' G(\mathbf{r}, \mathbf{r}'; i\omega + i\omega') W(\mathbf{r}, \mathbf{r}'; i\omega') \quad (3.37)$$

In FHI-aims, the self-energy is first calculated in the imaginary frequency grid, and then analytically continued to the real frequency grid. One popular way of performing the analytical continuation is to model the self-energy with a multi-pole expression [98], namely,

$$\Sigma(i\omega) \approx \sum_n \frac{A_n}{i\omega - B_n}, \quad (3.38)$$

where  $n$  is the number of poles, and  $A_n$  and  $B_n$  are complex numbers. Eq. (3.38) is used to fitted the calculated self-energy on imaginary axis using the non-linear least square fitting algorithm. Once the the parameters  $A_n$  and  $B_n$  are obtained that give the best fitting, the self-energy on the real frequency axis can be obtained by

$$\Sigma(\omega) \approx \sum_n \frac{A_n}{\omega - B_n}. \quad (3.39)$$

In practice  $n = 2$  (the so-called two-pole fitting) is often found to give the best performance.

---

## Tags for general section of `control.in`:

---

### Tag: `anacon_type` (`control.in`)

Usage: `anacon_type` value

Purpose: If set, specifies type of analytical continuation for the self-energy (we calculate the self-energy on the imaginary frequency axis, and hence need to continue it to the real axis)

- `value = 0` : The normal two-pole fitting
- `value = 1` : Pade approximation

Default: `value=0`

*Note* that the `anacon_type` only makes sense if `qpe_calc` or `sc_self_energy` is set, i.e., the post-processing-type self-energy calculation is required.

---

### Tag: `auxil_basis` (`control.in`)

Usage: `auxil_basis` type

Purpose: Specifies the type of auxiliary basis used in the "beyond-DFT" calculation.

`type` is a string, which can be set either as `full` or `opt`. Default: `full`. Here is a brief explanation.

- `full` : The auxiliary basis is constructed as the "on-site" pair products of the regular basis functions. The allowed pair products are controlled by the parameters `max_n_prodbas` and `max_l_prodbas` (see later). These pair products are then orthonormalized using Gram-Schmidt procedure for each atom.
  - `opt` : The auxiliary basis is obtained from an optimization procedure, and must be specified by hand in `control.in` – in the same spirit as the basis sets used in standard Gaussian-bases RI-MP2 calculations.
- 

### Tag: `default_prodbas_acc` (`control.in`)

Usage: `default_prodbas_acc` threshold

Purpose: Specifies the default for `prodbas_acc`

`threshold` is a real value, defining the onsite threshold for the auxiliary basis construction.

Default:  $10^{-4}$  for `RI_method` `lv1`, depends on species (`species_z`) otherwise.

See `prodbas_acc` for more details. Default settings are:

- $10^{-2}$  for  $Z \leq 10$  (light elements)

- $10^{-3}$  for  $10 < Z \leq 18$
- $10^{-4}$  for  $Z > 18$  (all heavier elements)

The old default (version 042811 and earlier) was simply  $10^{-2}$  for all elements. For light elements, this setting produces accurate total energies and energy differences to the sub-meV level in all our tests. For heavier elements, significant inaccuracies could happen in atomic total energies. These inaccuracies would cancel out in energy differences; to guarantee total energy accuracy as well, we now set significantly tighter defaults for `prodbas_acc` in this range (alas, also more expensive, both in time and memory use).

**Tag:** `default_max_l_prodbas` (`control.in`)

Usage: `default_max_l_prodbas` value

Purpose: Specifies the default for `max_l_prodbas`

Default: 20 for `RI_method` `lv1`, 5 otherwise.

**Tag:** `default_max_n_prodbas` (`control.in`)

Usage: `default_max_n_prodbas` value

Purpose: Specifies the default for `max_n_prodbas`

Default: 20 for `RI_method` `lv1`, 6 otherwise.

**Tag:** `RI_method` (`control.in`)

Usage: `RI_method` type

Purpose: Specifies the version of the resolution of identity used in the beyond-DFT calculations. Here `type` is a string, either having a value `svs` (for the “SVS” version, i.e., Eq. (3.35)) or `v` (for the “V” version, i.e. Eq. (3.34)). A `lv1` (for the “LVL” version).

Default: Non-periodic: `type=v`, which gives better accuracy for a given auxiliary basis. Periodic: `type=LVL_fast`, which provides the necessary reduction to what is essentially an  $O(N)$  framework.

Rules of thumb:

- `type=V` is the preferred method for non-periodic calculations. For formal reasons, this is clearly the most accurate version. However, neither a periodic version nor gradients (forces and relaxations) are implemented at present.
- `type=LVL_fast` is the preferred version for periodic calculations, as well as for non-periodic versions with forces and relaxation. It scales as  $O(N)$  and greatly limits the memory use compared to RI-V. `RI_method` `LVL_fast` localizes the expansion of the Coulomb potential of basis function products to two centers, as

described in Sec. 3.20. It also relies extensively on screening near-zero elements of the density matrix and of the Coulomb operator. The drawback, however, is that the localized version has slightly larger errors than RI-V for hybrid functionals, and significantly larger errors for correlated methods like MP2 or RPA. Forces have an additional drawback, in that they should only be used with *small* basis sets – for large basis sets, an ill-conditioning issue in the expansion is not yet caught in the present implementation. Therefore, treat forces as experimental. At present, use them but be very careful. We expect significant improvements in upcoming releases.

- `type=LVL_full` is a slower, non-screened version of LVL for non-periodic systems. Very useful as a reference to make sure.
- `type=SVS` is here only for testing purposes. This is the naive, purely overlap based version of RI and should not be used in production.

There is an additional option for non-periodic systems, `lv1_2nd`, which adds a correction term to the nonlocal exchange energy to make it “robust” in the Dunlap sense [26], that is, the error in the energy is quadratic in the error in the product expansion.

---

**Tag: `frequency_points`** (`control.in`)

Usage: `frequency_points` value

Purpose: If set, specifies the number of (imaginary) frequency points for the self-energy calculation.

Default: `value=80`.

*Note* that the `frequency_points` only makes sense if `qpe_calc` or `sc_self_energy` is set, i.e., the post-processing-type self-energy calculation is required.

---

**Tag: `time_points`** (`control.in`)

Usage: `time_points` value

Purpose: If set, specifies the number (imaginary) time points for the self-energy calculation.

Default: `value=80`.

*Note* that the `frequency_points` only makes sense if `qpe_calc` or `sc_self_energy` is set, i.e., the post-processing-type self-energy calculation is required.

---

**Tag: `hf_version`** (`control.in`)

Usage: `hf_version` version

Purpose: Allows to switch between the standard density-matrix based setup of the Fock operator, and an orbital based exchange operator.

`version` is a number, either 0 (alias `density_matrix`) or 1 (aliases `eigencoefficients` and `overlap`). Default: 0

The exchange operator can be constructed either by summing over all states *first* to construct the density matrix (`version` 0), or by a straightforward setup of orbitals and computation of the exchange operator only then (`version` 1).

Both versions are pure matrix algebra. For small systems (few states), `version` 1 is vastly more efficient, but towards large systems (many states) an efficiency crossover clearly favors the density matrix based update.

By default, FHI-aims relies on the density-matrix based update always, because this is more memory efficient, but for small systems (atoms) with lots of basis functions, this choice should be reconsidered. Please note that both versions should work seamlessly with Pulay mixing.

**Tag:** `maximum_frequency` (`control.in`)

Usage: `maximum_frequency` value

Purpose: If set, specifies the maximal (imaginary) frequency value for the self-energy calculation. The unit for `value` here is Hartree.

*Note* that the `maximum_frequency` only makes sense if `qpe_calc` or `sc_self_energy` is set, i.e., the post-processing-type self-energy calculation is required.

**Tag:** `maximum_time` (`control.in`)

Usage: `maximum_time` value

Purpose: If set, specifies the maximal (imaginary) time value for the self-consistent self-energy calculation. The unit for `value` here is Hartree<sup>-1</sup>.

*Note* that the `maximum_time` only makes sense if `sc_self_energy` is set, i.e., the post-processing-type self-consistent self-energy calculation is required.

**Tag:** `n_poles` (`control.in`)

Usage: `n_poles` value

Purpose: If set, specifies the number of poles (i.e. the number of functions of the form  $f_i(\omega) = 1/(b_i + i\omega)$ ) adopted in the pole-based computation of the Fourier transform in self-consistent *GW*-type calculations.

*Note* that the `n_poles` only makes sense if `sc_self_energy` is set, i.e., the post-processing-type self-consistent self-energy calculation is required.

---

**Tag:** `pole_max` (control.in)

Usage: `pole_max` value

Purpose: If set, specifies the position in the (imaginary) frequency axis of the largest poles (i.e. the largest  $b_i$  coefficient in  $f_i(\omega) = 1/(b_i + i\omega)$ ) used in computation of the Fourier transform in self-consistent GW-type calculations. The unit for value here is Hartree.

*Note* that the `pole_max` only makes sense if `sc_self_energy` is set, i.e., the post-processing-type self-consistent self-energy calculation is required.

---

**Tag:** `pole_min` (control.in)

Usage: `pole_min` value

Purpose: If set, specifies the position in the (imaginary) frequency axis of the smallest poles (i.e. the smallest  $b_i$  coefficient in  $f_i(\omega) = 1/(b_i + i\omega)$ ) used in computation of the Fourier transform in self-consistent GW-type calculations. The unit for value here is Hartree.

*Note* that the `pole_min` only makes sense if `sc_self_energy` is set, i.e., the post-processing-type self-consistent self-energy calculation is required.

---

**Tag:** `n_anacon_par` (control.in)

Usage: `n_anacon_par` value

Purpose: If set, specifies the number of (complex) parameters used for analytical continuation.

*Note* that the `n_anacon_par` only makes sense if `qpe_calc` or `sc_self_energy` is set, i.e., the post-processing-type self-energy calculation is required.

---

**Tag:** `prodbas_nb` (control.in)

Usage: `prodbas_nb` nb

Purpose: For very large scale beyond-GGA calculations, the distribution of auxiliary basis functions among the CPUs becomes problematic because each CPU only gets very few. The default Scalapack distribution is more tailored to efficient calculations and distributes these functions in chunks of finite size. In massively parallel runs, often each CPU gets either one or two of these chunks, leading to bad memory distribution. This can be circumvented, possibly sacrificing some performance, by setting the chunk size nb to "1".

nb is the chunk size of auxiliary basis functions.

Default:  $\min(16, \lfloor N_{\text{aux}}/N_{\text{proc}} \rfloor)$ .

**Tag:** `prodbas_threshold` (control.in)

Usage: `prodbas_threshold` threshold

Purpose: Prevent the possible ill-conditioning of the auxiliary basis, similar to `basis_threshold` for the regular basis.

threshold is a small positive real number for the eigenvalue of the Coulomb matrix for the auxiliary basis. Default:  $10^{-5}$ .

The the auxiliary basis functions centered on different atoms are nonorthogonal, and the possible linear dependence (with certain accuracy) between them and the resultant behavior has to be carefully eliminated. This is achieved by setting the cutoff threshold for the auxiliary basis `prodbas_threshold`. From many test calculations, it is found that the reliable value for threshold here are between  $10^{-5}$  to  $10^{-3}$ . Within this window, the total energy may still have some noticeable change, but the energy difference is usually negligible. It is suggested that the user should play with this value if he/she is not sure about his/her result.

---

**Tag:** `sbtgrid_lnrange` (control.in)

Usage: `sbtgrid_lnrange` lnrange

Purpose: for `use_logsbt`, set the range of the logarithmic grid (in logarithmic units). Default is 45, which corresponds to nearly twenty orders of magnitude. Please note that the range should be larger than intuitively guessed because the range is the same both in real and reciprocal space (for algorithmic reasons) and the tails in reciprocal space have to be captured.

---

**Tag:** `sbtgrid_lnr0` (control.in)

Usage: `sbtgrid_lnr0` lnr0

Purpose: for `use_logsbt`, set the onset of the logarithmic grid in real space. The default is  $-38$ .

---

**Tag:** `sbtgrid_lnk0` (control.in)

Usage: `sbtgrid_lnk0` lnk0

Purpose: for `use_logsbt`, set the onset of the logarithmic grid in Fourier space. The default is  $-25$ .

---

**Tag:** `sbtgrid_N` (control.in)

Usage: `sbtgrid_N` N

Purpose: for `use_logsbt`, set the number of logarithmic grid points both in real and Fourier space. The default is 1024.

For the accuracy, the density of points  $N/\lnrange$  is relevant. Additionally, one has to make sure that the tails in real and Fourier space are properly included.

---

**Tag:** `state_lower_limit` (control.in)

Usage: `state_lower_limit` value

Purpose: If set, specifies the lowest single-particle eigenstate to be included for the quasiparticle calculation.

*Note* that the `state_lower_limit` only makes sense if `qpe_calc` is set, i.e., the post-processing-type self-energy calculation is required.

---

**Tag:** `state_upper_limit` (control.in)

Usage: `state_upper_limit` value

Purpose: If set, specifies the highest single-particle eigenstate to be included for the quasiparticle calculation.

*Note* that the `state_upper_limit` only makes sense if `qpe_calc` is set, i.e., the post-processing-type self-energy calculation is required.

---

**Tag:** `use_logsbt` (control.in)

Usage: `use_logsbt` bool

Purpose: if set, the two-center integrals are calculated by one-dimensional integrations in Fourier-space. Otherwise, a three-dimensional integration on a grid is performed. This feature is so far experimental and disabled by default. But in principle, it should be much faster and more accurate. So far, only in effect for Hartree-Fock and hybrid exchange calculations.

The algorithm for the overlap and Coulomb integrals is described by Talman in [104]. It uses an efficient spherical Bessel transform on a logarithmic radial grid [105, 40, 41] to obtain the Fourier transform of the auxiliary basis functions and calculates the integrals in Fourier space.

---

**Tag:** `use_ovlp_swap` (control.in)

Usage: `use_ovlp_swap`

Purpose: if set, the atomic orbital (AO) based 3-index overlap matrix ("ovlp\_3fn" in the source code) is written to the disk before transforming to the molecular orbital (MO) based 3-index overlap matrix ("O\_2bs1HF" for HF calculations and "ovlp\_3KS" for GW calculations in the source code). This avoids the double allocation of both the AO-based 3-index integral matrix and MO-based 3-index integral matrix at the same time and thus reduces the memory cost by about a factor of two.

## Subtags for *species* tag in `control.in`:

### **species** sub-tag: `aux_gaussian` (`control.in`)

Usage: `aux_gaussian` L N [alpha]  
       [ alpha\_1 coeff\_1 ]  
       [ alpha\_2 coeff\_2 ]  
       [ ... ]  
       [ alpha\_N coeff\_N ]

Purpose: For `auxil_basis` opt, adds a Gaussian basis function to the auxiliary basis set for the Coulomb operator.

L is an integer number, specifying the angular momentum

N is an integer number, specifying how many primitive Gaussians comprise the present radial function

alpha : If N=1, this is the exponent defining a primitive Gaussian function [in bohr<sup>-2</sup>].

alpha\_i coeff\_i : If N>1,  $i = 1, \dots, N$  additional lines specify exponents  $\alpha_i$  and expansion coefficients  $g_i$  for a non-primitive linear combination of Gaussians.

See the description of `gaussian` basis functions; Gaussian basis functions in the *auxiliary* basis use essentially the same infrastructure.

### **species** sub-tag: `for_aux` (`control.in`)

Usage: `for_aux` basis options

Purpose: Add a extra basis function to constructor of auxiliary basis function.

Basis is a either `hydro` or `ionic` basis function keyword and options are options for that basis function.

Experimental! These extra basis function are used ONLY when auxiliary basis set is constructed by adding keyword `for_aux` before normal basis function string in species details. Currently support only `hydro` and `ionic` basis functions. See the details of `hydro` and `ionic` for exact details of the options for basis functions. This feature is still experimental and has not been utterly tested.

### **species** sub-tag: `prodbas_acc` (`control.in`)

Usage: `prodbas_acc` threshold

Purpose: Technical cutoff criterion for on-site orthonormalization of auxiliary radial functions. Here `threshold` is a small positive real value. Default: See below.

To construct the set of auxiliary basis functions, the radial functions for a single species are "on-site" Gram-Schmidt orthonormalized. If the norm of the function after orthonormalization is smaller than `threshold`, that function is omitted.

The present default values are:

- $10^{-2}$  for  $Z \leq 10$  (light elements)
- $10^{-3}$  for  $10 < Z \leq 18$
- $10^{-4}$  for  $Z > 18$  (all heavier elements)

The old default (version 042811 and earlier) was simply  $10^{-2}$  for all elements. For light elements, this setting produces accurate total energies and energy differences to the sub-meV level in all our tests. For heavier elements, significant inaccuracies could happen in atomic total energies. These inaccuracies would cancel out in energy differences; to guarantee total energy accuracy as well, we now set significantly tighter defaults for `prodbas_acc` in this range (alas, also more expensive, both in time and memory use).

For simplicity, it is also possible to use `default_prodbas_acc` to set a global value of `prodbas_acc` across all elements.

*Note* that the `prodbas_acc` should not be confused with the `prodbas_threshold`. The former is used when constructing the auxiliary basis functions for each species, whereas the latter is used to deal with the ill-conditioning behavior of the Coulomb repulsion and/or the overlap matrix between the set of auxiliary basis functions for the whole systems.

---

**species sub-tag:** `max_n_prodbas` (control.in)

Usage: `max_n_prodbas` value

Purpose: Specifies the maximal principal quantum number for the *regular* basis function to be included in the *auxiliary (product)* basis construction.

value is a positive integer number here.

*Note* that `max_n_prodbas` has an effect only when `auxil_basis` is setted to full.

---

**species sub-tag:** `max_l_prodbas` (control.in)

Usage: `max_l_prodbas` value

Purpose: Specifies the maximal angular quantum number for the *auxiliary (product)* basis function. Any possible auxiliary basis with an angular momentum higher than `max_l_prodbas` is excluded.

value is a positive integer number here.

*Note* that `max_l_prodbas` controls the *auxiliary* basis whereas `max_n_prodbas` controls only directly the *regular* basis. `max_l_prodbas` has an effect regardless whether `auxil_basis` is setted to full or opt.

## 3.20 Periodic Hartree-Fock and hybrid functionals

Periodic versions of the Hartree-Fock method and of hybrid density functionals have been implemented in FHI-aims recently. For single-point calculations (no geometry relaxation), our experience is very good so far. Still, our implementation is new, so please be careful with the obtained results. Benchmark them carefully and consult the developers when in doubt.

There are three specific issues from a usability point of view which should be considered:

- For geometry relaxation, the present version should only be used with caution. Forces appear to work for *tier1* type basis sets, but for large basis sets, failures may occur because an ill-conditioning issue in the forces is not yet caught. This functionality is simply still experimental.
- The `RI_method` described below is slightly less accurate (for formal reasons) than the standard for non-periodic systems, RI-V. Please bear this in mind. For standard solids and hybrid functionals, the effects seem very small, but for larger molecules and small basis sets, the effects become measurable.
- For band structure output, the `exx_band_structure_version` keyword allows to toggle between a slow but safe, reciprocal-space based version, and a faster real-space version that works only when a relatively dense  $k$ -space grid has been used during the regular s.c.f. cycle. The underlying reason is that the real-space Born-von Karman cell of the regular s.c.f. cycle may become too small to accommodate some  $k$ -vectors that are not exact reciprocal lattice vectors of the Born-von Karman cell.

In periodic Hartree-Fock (and hybrid functional) implementations, the key quantity that needs to be evaluated is the exact-exchange matrix,

$$K_{ij}(\mathbf{k}) = \sum_{kl, \mathbf{q}} D_{kl}(\mathbf{q}) \iint d\mathbf{r} d\mathbf{r}' \frac{\phi_{i\mathbf{k}}^*(\mathbf{r}) \phi_{k\mathbf{q}}(\mathbf{r}) \phi_{l\mathbf{q}}^*(\mathbf{r}') \phi_{j\mathbf{k}}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} \quad (3.40)$$

where  $\mathbf{k}, \mathbf{q}$  are the Bloch vectors,  $\phi_{i\mathbf{k}}(\mathbf{r})$  is the Bloch summation of the  $i$ -th atomic orbital  $\phi_i(\mathbf{r} - \mathbf{R})$  living in the unit cell  $\mathbf{R}$ , and  $D_{kl}(\mathbf{q})$  is the density matrix.  $K_{ij}(\mathbf{k})$  can be obtained from its Fourier transform,

$$K_{ij}(\mathbf{k}) = \sum_{\mathbf{R}} e^{i\mathbf{k}\mathbf{R}} X_{ij}(\mathbf{R}), \quad (3.41)$$

where

$$X_{ij}(\mathbf{R}) = \sum_{kl} \sum_{\mathbf{R}'} D_{kl}(\mathbf{R}') \sum_{\mathbf{R}''} \iint d\mathbf{r} d\mathbf{r}' \frac{\phi_i(\mathbf{r}) \phi_k(\mathbf{r} + \mathbf{R}'') \phi_j(\mathbf{r}' + \mathbf{R}) \phi_l(\mathbf{r}' + \mathbf{R}' + \mathbf{R}'')}{|\mathbf{r} - \mathbf{r}'|} \quad (3.42)$$

In FHI-aims, periodic Hartree-Fock and hybrid density functionals are implemented in two different ways. One implementation is based on the “k-space” formulation, where

one computes  $K_{ij}(\mathbf{k})$  directly from Eq. (3.40). An alternative, and more efficient implementation is based on the “real-space” formulation, where one first computes  $X_{ij}(\mathbf{R})$  from Eq. (3.42), and then Fourier transform it to  $K_{ij}(\mathbf{k})$ . The “real-space” implementation is used in the code by default, and the “k-space” implementation is only used for crossing-check purposes.

Both implementations are based on a localized resolution-of-identity approximation, which we termed as “RI-LVL”, in analogy to “RI-SVS” and “RI-V” introduced in Sec. 3.19. Under “RI-LVL”, the products of two normal basis functions  $(i, j)$  centering at atoms  $A_i$  and  $A_j$  are expanded only in terms of auxiliary functions centering on these two atoms. Possible contributions of auxiliary functions from a third center are excluded in this approximation, in contrast to “RI-V”. Specifically, one has

$$\phi_i(\mathbf{r} - A_i)\phi_j(\mathbf{r} - A_j) = \sum_{\mu} C_{i(A_i),j(A_j)}^{\mu(A_i)} P_{\mu}(\mathbf{r} - A_i) + \sum_{\nu} C_{i(A_i),j(A_j)}^{\nu(A_j)} P_{\nu}(\mathbf{r} - A_j), \quad (3.43)$$

where  $\mu$  and  $\nu$  enumerate the auxiliary basis functions centering on atom  $A_i$  and  $A_j$  respectively. This approximation has been extensively benchmarked with respect to the more accurate “RI-V” approximation for finite systems, and with respect to other independent implementations for molecular systems. The achieved accuracy is remarkable and should be sufficiently good for production calculations.

Periodic Hartree-Fock and hybrid-functional calculations can be run in the same manner as the periodic LDA and GGA cases, by setting the keyword `xc` to `hf` or desired hybrid functionals, and setting the `k_grid` mesh to appropriate values. As mentioned above, by default the “real-space” periodic Hartree-Fock implementation will be invoked. There are two thresholding parameters (detailed below) which control the balance between the computational load and accuracy in the calculation. One may also switch to the “k-space” implementation of periodic Hartree-Fock and hybrid functionals for testing or comparison purposes by setting the keyword `use_hf_kspace` to be true (see below). The thresholding parameters do not apply to the “k-space” implementation, however.

## Tags for general section of `control.in`:

---

### Tag: `coulomb_threshold` (`control.in`)

Usage: `periodic_hf coulomb_threshold` value

Purpose: This sets a threshold value for a key ingredient in the construction of the exact-exchange matrix – the Coulomb matrix. The Coulomb matrix elements below the specified threshold value are discarded in the calculation. Suggested values are between  $10^{-6}$  and 0. The default value is  $10^{-10}$ .

---

### Tag: `exx_band_structure_version` (`control.in`)

Usage: `exx_band_structure_version` value

Purpose: A periodic band structure calculation can be performed either using a real-space version (`value=1`) or a reciprocal-space version (`value=2`). No default – user must decide.

`value` is an integer, either 1 or 2.

If `output band` is requested for a periodic Hartree-Fock or hybrid functional calculation, adhere to the following rules:

- Do not use excessively many  $k$  points in each band segment, for instance no more than 20.
- The real-space band structure version `value=1` is fast and accurate IF a reasonably dense `k_grid` has been used during the s.c.f. calculation. We recommend to test to be sure.
- If the plotted band structure from the real-space version `value=1` has obvious numerical problems, please switch to either a denser `k_grid` during s.c.f., or to `exx_band_structure_version 2`. The latter will always work, as the critical part of the work is handled in reciprocal space. As a consequence, though, sparsity in real space can no longer be exploited, and the band structure calculation becomes much slower than the real-space version.
- In case of doubt, the band structure ONLY at  $k$ -points used during the s.c.f. cycle itself can also be printed along certain directions by using the `output band_during_scf` keyword, which ensures that only the information that went into the s.c.f. cycle is actually used. This is mainly useful for debugging purposes but since the functionality in question is new, it may be helpful.

We apologize that this decision process is still a bit rough around the edges and leaves an essential decision up to the user. However, consider this: The functionality in question is new, and the above procedure provides a simple way to make band structure output work, and work safely. It is now generally not a problem to produce band structures with hybrid functionals in FHI-aims, if one follows the above hints.

---

**Tag:** `screening_threshold` (control.in)

Usage: `periodic_hf screening_threshold` value

Purpose: This sets a screening parameter in a periodic Hartree-Fock (or hybrid functional) calculation. The real-space exact-exchange matrix elements below the specified threshold value are neglected in the calculation. Suggested values are between  $10^{-6}$  and 0. Smaller values mean better accuracy but heavier computational loads. The default value is  $10^{-7}$ .

---

**Tag:** `use_hf_kspace` (control.in)

Usage: `use_hf_kspace` flag

Purpose: The “k-space” periodic HF implementation can be invoked by setting flag to be `.true`.

## 3.21 Large-scale, massively parallel: Memory use, sparsity, communication, etc.

In one way or another, most options available with FHI-aims concern physical algorithms or numerical choices, including those affecting the accuracy and/or efficiency of a given task. As much as possible, FHI-aims attempts to use the exact same code and settings to describe any given system, and on any kind of computer hardware. Usually, this guarantees efficient code on all ends, and improvements for one class of systems immediately benefit all others

However, for very large tasks [meaning here several hundred up to thousands of atoms, depending on the system] and/or on parallel architectures with possibly (ten)thousands of processors, memory and communication constraints may come into play that require specific workarounds not needed (or beneficial) in normal systems. On a practical level, FHI-aims attempts to be as memory-parallel as possible without loss of efficiency. However, if any such modification could affect the performance for normal systems and computer architecture adversely, we recommend to switch it on separately only when needed.

On massively parallel machines, and for very large problems, the following options are particularly important:

- If ScaLapack is used (`KS_method scalapack`), Keyword `collect_eigenvectors` allows to switch off the collection of the full eigenvectors  $c_{il}$  on each CPU, saving both communication time and a significant amount of memory.
- In the cluster case, keyword `use_density_matrix` should be used for system sizes of a few hundred atoms and above. (This is the default for all periodic systems anyway).
- For the cluster case, and if ScaLapack is used, keyword `packed_matrix_format` may save a significant amount of memory in the construction of the overlap, Hamiltonian, and density matrices. (This is the default for all periodic systems anyway).
- For the cluster case with more than 200 atoms, the non-periodic Ewald method can be used to accelerate the calculation, see section 3.7.1.
- Keyword `use_local_index`, which ensures that integration grid batches on the same CPU are always close together, requiring only a subset of the packed Hamiltonian matrix as working space for integrals on each CPU. Use `load_balancing` to improve load-balancing in this case to avoid the a negative impact on the efficiency.
- The keyword `distributed_spline_storage` avoids to store the complete multipolar decomposition of the charge density on each processor. Instead, they are only stored for those atoms with local grid points in reach.
- For beyond-GGA: Keyword `prodbas_nb` can be used to enhance the distribution of memory intensive arrays, possibly sacrificing performance.

Finally: There is a new keyword, `use_alltoall`, that allows to switch the communication behaviour of FHI-aims for very large runs when parallel linear algebra (scalapack / ELPA) is used. The default switching point is set at 1024 MPI tasks and affects CPU time (very many cores) and memory use. If you see changes around 1024 cores (especially lack of memory in “normal” LDA/GGA calculations), see there.

## Tags for general section of `control.in`:

---

### Tag: `collect_eigenvectors` (`control.in`)

Usage: `collect_eigenvectors` flag

Purpose: When ScaLapack is used, allows to switch off the collection of all eigenvectors to each CPU, saving memory and computation time.

Restriction: Eigenvectors are needed for some post-processing functionality, e.g., in order to obtain a full Mulliken analysis or a partial density of states.

flag is a logical string, either `.false.` or `.true.` Default: `.false.`

---

### Tag: `distributed_spline_storage` (`control.in`)

Usage: `distributed_spline_storage` flag

Purpose: Request to store the multipolar decomposition of the density only for the atoms needed on a given processor for the corresponding parts of the Hartree potential.

flag is a logical string, either `.false.` or `.true.` Default: `.false.`

This flag is `.false.` by default because the complete splined density might be needed for some postprocessing or output option.

---

### Tag: `force_mpi_virtual_topo` (`control.in`)

Usage: `force_mpi_virtual_topo` flag

Purpose: Auxiliary option to try to force the MPI library to respect the topology of the nodes used (several tasks within each shared-memory node vs. slower communication between different nodes)

flag is a logical string, either `.false.` or `.true.` Default: `.false.`

If requested, enables to cache the (deduced) topology of the nodes to the default communicator. In principle, the communication layer should then obtain more information on the network topology and organize the communication pattern more efficiently. However, nothing is guaranteed. The standard does not force MPI to respect the cached information, and in all decent MPI library implementations, this information should already be provided by the system.

In short: Perhaps try this if a truly strange communication pattern is observed, but probably, there will be no effect.

---

### Tag: `load_balancing` (`control.in`)

Usage: `load_balancing` flag

Purpose: Using the keyword `use_local_index` has a negative impact on the load-balancing of grid operations. With `load_balancing .true.`, this efficiency drawback can be avoided by explicit reassignment of grid point batches to processors according to timings of test runs.

flag is a logical string, either `.false.` or `.true.` Default: `.false.`

Thanks to the addition of this feature by Rainer Johanni, using `use_local_index` no longer has any negative impact on performance and should always be used when memory is an issue.

---

**Tag:** `packed_matrix_format` (control.in)

Usage: `packed_matrix_format` type

Purpose: Allows to use a packed-matrix format for the Hamiltonian, overlap, and density matrices of the real-space basis functions.

type is a string that indicates the type of packing used. Default: `none` for the cluster case, `index` for periodic geometries.

The following options exist for type:

- `none` - no packing is used
- `index` - matrices are packed by strictly eliminating *all* near-zero elements of all three matrices. Elements  $ij$  are eliminated if *both* the overlap matrix element and the initial Hamiltonian matrix element are smaller than a threshold set by keyword `packed_matrix_threshold`.

Packing the overlap, Hamiltonian and density matrices reduces the size of these arrays during matrix integration, at the expense of some small effort to correctly sort intermediate results during the integration appropriately.

From a technical point of view, packing only makes sense if the full Hamiltonian is not required later anyway, during the eigenvalue solution. This is the case:

- In the cluster case: if ScaLapack is used (`KS_method scalapack`).
  - In the periodic case: for more than one k-point, and/or if ScaLapack is used.
- 

**Tag:** `packed_matrix_threshold` (control.in)

Usage: `packed_matrix_threshold` tolerance

Purpose: Tolerance value below which the elements of the overlap / Hamiltonian matrices are eliminated from the `packed_matrix_format`.

tolerance : A small positive real numerical value. Default:  $10^{-13}$ .

---

**Tag:** `prune_basis_once` (control.in)

Usage: `prune_basis_once` flag

Purpose: Stores the indices of the non-zero basis functions for each integration batch in memory

flag is a logical string, either `.false.` or `.true.` Default: `.true.`

All operations for the integrations and the electron density update are  $O(N)$  operations, but verifying which basis functions are non-zero for each integration batch requires to check each basis function, and is thus an  $O(N^2)$  operation with a small prefactor. This step can be avoided by checking for the non-zero basis functions once, and then storing their indices in memory for each batch of integration points. For very large systems and restricted memory, it may be worth trying to switch this feature off to save some memory, otherwise this should always be done.

---

**Tag:** `store_EV_to_disk_in_relaxation` (control.in)

Usage: `store_EV_to_disk_in_relaxation` flag

Purpose: During relaxation, eigenvectors from a previous geometry can be stored to disk instead of in memory in case the next step is reverted.

flag is a logical string, either `.false.` or `.true.` Default: `.false.`

During relaxation (see `relax_geometry`), geometry steps can be rejected if the total energy increased unexpectedly. In order to revert to a previous step, it is necessary to access the Kohn-Sham eigenvectors used to initialize that step, which must hence be stored for that purpose. In normal calculations, eigenvector storage is not a problem, but their size grows as  $O(N^2)$  with system size. For very large system sizes, their storage in memory can become a bottleneck, which is here circumvented by the option to store them to disk.

This keyword is *not* related to the restart functionality of the `restart` keyword, since it is an old, not a current, set of Kohn-Sham eigenvectors that may be needed when reverting a geometry step.

---

**Tag:** `use_2d_corr` (control.in)

Usage: `use_2d_corr` flag

Purpose: Allows to switch on or off the two-dimensional distribution of data structures for correlated methods.

flag is a logical string, either `.false.` or `.true.` Default: `.true..`

Only relevant for correlated beyond-hybrid methods.

---

**Tag:** `use_alltoall` (control.in)

Usage: `use_alltoall` flag

Purpose: Allows to switch some communication calls in FHI-aims, depending on the number of tasks.

`flag` is a logical string, either `.false.` or `.true.` Default: `.false.` below 1024 MPI tasks, true for 1024 and above.

Only relevant for parallel linear algebra (use of scalapack or ELPA).

When running on a system with many CPUs, it can be much faster to use “all-to-all” communication (`mpi_alltoallv`) than doing `n_tasks` times a `sendrecv` call; this is, for example, noticeable on the BlueGene/P with  $\approx 10^5$  MPI tasks at a time. For much fewer tasks, the effect is not usually relevant.

On the other hand, “all-to-all” costs a significant amount of memory, and this memory pressure will be felt for much fewer MPI tasks (e.g., Intel architecture with hundreds of CPU cores at a time).

Thus we use “sendrecv” (individual communication) to using `mpi_alltoallv` when using 1024 CPUs or more. `use_alltoall` can be employed to enforce one or the other type of call throughout. If you see too much MPI-related memory use at 1024 tasks or above, try it—and let us know.

---

**Tag:** `use_local_index` (`control.in`)

Usage: `use_local_index` flag

Purpose: Reduces work space size for Hamiltonian / overlap matrix during integrals by storing only those parts that are touched by any grid points assigned to the present CPU.

Restriction: Supported for standard LDA and GGA with ScaLapack and packed matrices, but not for some non-standard options. Requires `prune_basis_once` and a parallel `grid_partitioning_method`.

`flag` is a logical string, either `.false.` or `.true.` (or `if_scalapack`. Default: `.false.`

For very large systems with many CPUs, it makes sense to assign only integration grid batches close to one another to the same CPU, and to store only the Hamiltonian / overlap matrix parts needed for those batches on that CPU. However, the subsequent merge of all results into the ScaLapack infrastructure then becomes more difficult, and some overhead results from the changed load balancing. This option is therefore not used by a standard call to FHI-aims, but must be switched on explicitly if needed.

---

**Tag:** `walltime` (`control.in`)

Usage: `walltime` seconds

Purpose: Can limit the wall clock time spent by FHI-aims explicitly, e.g., to obtain the correct final output before a queuing system shuts down a calculation.

`seconds` is the integer requested wall clock time in seconds. Default: no limit.

In order to reach the postprocessing phase and write information required at the end of a calculation in an organized manner, FHI-aims can force a stop before a certain amount of real time (wall clock time) is exceeded. In order to achieve this safely, FHI-aims uses an internal estimate of the duration of a single s.c.f. iteration within the calculation, and stops if it estimates that the next iteration will take more time than what remains available.

## 3.22 Output options

The primary (and most important) output of FHI-aims is written to the standard output channel, and can / should be captured in a file from there. However, FHI-aims provides a host of further output options that can be activated to yield more specialized data not ordinarily required from a standard calculation, but highly useful for specific purposes.

The majority of these output options is activated by invoking the `output` option in file `control.in`. The individual subkeywords to this keyword are therefore listed separately, towards the end of this section. In addition, some particularly important output options are revisited with examples in Chapter 4.

## Tags for `geometry.in`:

`verbatim_writeout` Usage: `verbatim_writeout` flag

Purpose: Enables or suppresses the writing of `geometry.in` to the FHI-aims standard output stream exactly as it is read the first time.

`flag` is a logical variable (`.true.` or `.false.`). Default: `.true.` .

By default, `geometry.in` is now written (copied) verbatim into the FHI-aims standard output as it is parsed for the first time, allowing to reproduce exactly any FHI-aims calculation simply by copy-pasting that part to a new `geometry.in` file.

If `verbatim_writeout` is set to false anywhere in `geometry.in`, no writing will occur from that point on forward.

The exact same option (same keyword / syntax) can also be used in `control.in`, producing the same effect there.

Note that the keyword has the same name in `geometry.in` and `control.in`, and is therefore only documented as a clickable link for `control.in`. Apologies for this omission.

## Tags for general section of `control.in`:

### Tag: `dos_kgrid_factors` (`control.in`)

Usage: `dos_kgrid_factors`  $n_1 n_2 n_3$

Purpose: If set, a post-scf density of states is computed with a denser  $k$ -point grid than used in the s.c.f. cycle.

Only useful in conjunction with the keyword `output dos`, and only for periodic systems.

In a periodic calculation, one usually specifies the basic `k_grid` used to obtain the self-consistent electron density, total energy etc. Such  $k$ -space grids are usually fairly sparse, and if a density of states (DOS) is calculated directly from the eigenvalues stored at these  $k$ -points only, the DOS will either look choppy, or (after significant broadening), smooth, broad, and blurred.

A simple remedy is to use the original `k_grid` while approaching self-consistency as usual, but then compute the DOS using an auxiliary  $k$ -grid that is made *denser* by factors  $n_1$ ,  $n_2$ ,  $n_3$ , respectively. For example, the settings

```
k_grid 10 10 10
```

```
output dos [...]
```

```
dos_kgrid_factors 8 8 8
```

mean that the s.c.f.-cycle itself is run with a  $10 \times 10 \times 10$   $k$ -point grid, but subsequently, a density of states is computed with an  $80 \times 80 \times 80$   $k$ -point grid.

### Tag: `evaluate_work_function` (`control.in`)

Usage: `evaluate_work_function`

Purpose: Surface slab calculations only – if true, the work functions of both slab surfaces will be evaluated.

This option requires that a reference  $z$  coordinate for the electrostatic potential evaluation deep in the vacuum be provided by hand, through the keyword `set_vacuum_level`. The surface must be parallel to the  $xy$  plane.

The output for the “upper” and “lower” surface of the slab (larger and smaller  $z$  value, respectively), will be printed separately. Note that for *non-symmetric* slabs, these work functions should generally be different. In practice, this behavior is correctly reproduced only if the `use_dipole_correction` is additionally specified.

Specifically, the reference Hartree potential component for the work function evaluation is only the long-range (reciprocal-space) Hartree potential term of the Ewald sum, not the full electrostatic potential. Thus, the vacuum level *must* be specified in a region where all real-space components of the electrostatic potential have safely died away to zero. You may achieve this by increasing the vacuum layer thickness, which can be done at very small overhead cost in FHI-aims.

---

**Tag:** `output` (`control.in`)

Usage: `output` type [further options]

Purpose: This is the central keyword that controls most of the non-standard output that can be written by FHI-aims.

type is a string that specifies the kind of requested output; any further needed options, or possibly additional lines, depend on type.

The list of additional output types is given as a separate subsection below.

---

**Tag:** `output_in_original_unit_cell` (`control.in`)

Usage: `output_in_original_unit_cell` flag

Purpose: Shifts the atoms in a periodic calculation back into the first unit cell before printing them out at the beginning of a new geometry step.

flag is a logical string, either `.true.` or `.false.` Default: `.true.`

In some, atoms in FHI-aims can unexpectedly “switch” unit cells during relaxation. This has no effect on the calculation, but the output geometry coordinates (written to the standard output stream) will look strangely detached when visualized. By default, FHI-aims maps its coordinates back to the first unit cell anyway, but this behavior can be forcibly switched off if so requested (makes nicer movies).

---

**Tag:** `output_level` (`control.in`)

Usage: `output_level` level

Purpose: Allows to increase the amount of output written to the standard output of FHI-aims.

level is a string that determines the amount of output written. Default: `normal`.

If increased to `full`, the Kohn-Sham eigenvalues of every s.c.f. iteration are written to the standard output file. For single-point calculations, this may be quite desirable, but leads to unmanageable file sizes for long relaxation or molecular dynamics runs.

Another option, useful for long molecular dynamics (MD) runs, is `MD_light`. It writes standard output only in the initialization part and at the end of each MD step, while a minimal output is written for the single scf cycle.

---

**Tag:** `verbatim_writeout` (`control.in`)

Usage: `verbatim_writeout` flag

Purpose: Enables or suppresses the writing of `control.in` to the FHI-aims standard output stream exactly as it is read the first time.

flag is a logical variable (`.true.` or `.false.`). Default: `.true.`.

By default, `control.in` is now written (copied) verbatim into the FHI-aims standard output as it is parsed for the first time, allowing to reproduce exactly any FHI-aims calculation simply by copy-pasting that part to a new `control.in` file.

If `verbatim_writeout` is set to `false` anywhere in `control.in`, no writing will occur from that point on forward.

The exact same option (same keyword / syntax) can also be used in `geometry.in`, producing the same effect there.

## Specific options output keyword:

---

### output sub-tag: `aitranss` (`control.in`)

Usage: `output aitranss`

Purpose: Writes Kohn-Sham eigenvectors  $c_{il}$  and energies  $\varepsilon_l$  (where  $i$  is a basis function index and  $l$  is an eigenstate index) of each spin channel and overlap matrix  $s_{ij}$  into separate ASCII-files in a format compatible with AITRANSS (*ab initio* transport simulations) package.

Restrictions: This functionality is available only for *non-periodic* systems. If `KS_method scalapack` is used, `packed_matrix_format` is not supported. Flag `use_local_index` is not supported either.

Please, look at Chapter 5 for a comprehensive description on how to perform transport simulations across nanoscale objects.

---

### output sub-tag: `atom_proj_dos` (`control.in`)

Usage: `output atom_proj_dos` `Estart` `Eend` `n_points` `broadening`

Purpose: Writes an atom-projected, angular-momentum resolved partial density of states (pDOS).

`Estart` : Lower bound of the single-particle energy range for which the pDOS are given.

`Eend` : Upper bound of the single-particle energy range for which the pDOS are given.

`n_points` : Number of energy data points for which the pDOS are given.

`broadening` : Gaussian broadening applied to obtain a smooth partial density of states based on the peaks produced by individual states.

This option is based on a Mulliken Analysis and shares its syntax with `output dos` and `output species_proj_dos`. See also section 4.4 for more details.

There are two types of output files for each atom:

- `atom_proj_dos_number_raw.dat`, where *number* denotes the atom number in the order of `geometry.in`. This file contains the total and angular-momentum resolved DOS components as a function of eigenvalue energy (first column) as used internally in FHI-aims. The energy zero is then given by the vacuum level (non-periodic systems) or by the  $G=0$  component of the long-range Hartree potential (periodic systems).
- `atom_projected_dos_number.dat`, which gives the same information, except that the energy zero is shifted to the Fermi energy (metallic systems) or valence band maximum (insulators), respectively.

Note that projected densities of states such as given here must be based on some kind

of projection orbitals, the choice of which is somewhat arbitrary by necessity. This is thus a tool for *qualitative* analyses.

In FHI-aims, we project directly on the atom-centered angular-momentum components as defined by the *overlapping* basis set. This definition becomes the more arbitrary the larger the basis set, just like a [mulliken](#) analysis. The closer the full basis comes to completeness, the more ambiguous will a [mulliken](#)-like analysis become, since it may not be *a priori* clear which electrons should be counted towards the basis functions of one atom vs. those of another atom. Thus, do *not* expect a pDOS to simply converge as the basis set size is increased; use it as a qualitative indicator of trends, but nothing more.

**output sub-tag: band** (control.in)

Usage: `output band kstart1 kstart2 kstart3 kend1 kend2 kend3 n_points name_start name_end`

Purpose: Plots a band along the line from  $\langle k_{start1}, k_{start2}, k_{start3} \rangle$  to  $\langle k_{end1}, k_{end2}, k_{end3} \rangle$  at `n_points` equally spaced points. The  $k$ -vectors are written in relative coordinates of the reciprocal basis vectors.

Several bands can be plotted; FHI-aims outputs one file per specified `output band` line. These files are named `band1XXX.out` and have the format

```
ipoint k1 k2 k3 E1 occ1 E2 occ2 ... EN occN,
```

i.e. they specify not only the bands for each  $k$ -point but also the occupation number for this particular point. Since this format contains all the important information, but is not particularly useful for actually plotting the band structure, we provide a small script which is described in section 4.4.

Note: the last two input options are technically not needed by the FHI-aims main program, but they are seriously helpful when turning this data into a plot and are used by the band plotting script provided along with this distribution, see Section 4.4 for details.

For periodic band structure output, the `exx_band_structure_version` keyword must be set – see the respective Section ?? for a brief explanation of the background.

**output sub-tag: band\_during\_scf** (control.in)

Usage: `output band_during_scf kstart1 kstart2 kstart3 kend1 kend2 kend3 name_start name_end`

Purpose: Plots a band along the line from  $\langle k_{start1}, k_{start2}, k_{start3} \rangle$  to  $\langle k_{end1}, k_{end2}, k_{end3} \rangle$  but only at those  $k$  points that are already part of the normal s.c.f. [k\\_grid](#). The  $k$ -vectors are written in relative coordinates of the reciprocal basis vectors.

This keyword allows to get the band structure along a certain reciprocal-space direction, but *only* at those  $k$ -points that are already used during the s.c.f. calculation. If there are no appropriate  $k$ -points, no band structure is printed. If there are appropriate  $k$ -points,

they are printed in the same format as the normal band structure from `outputband`, although some additional editing may be required to get a clean plot.

The purpose of this keyword is to allow to extract a band structure even in cases when the normal `outputband` functionality is experimental or, for some reason, not available.

**output sub-tag: basis** (control.in)

Usage: `output basis`

Purpose: Writes radial functions before and after orthonormalization, as well as second derivatives and the basis-defining potentials to separate files.

This output option allows to visualize the basis functions used, as well as some of the other defining pieces of the basis set. Note that the output is written for each grid point of the dense 1-dimensional `logarithmic` grid, with the radius given in bohr.

Specifically, this option produces the following types of files:

- `Ai_j_nl_base.dat` : Atomic (minimal basis) radial function  $u(r)$  after the basis-confining potential was applied, for `species` number  $i$ , atomic-like (minimal) radial function number  $j$ , radial and angular quantum numbers  $nl$ .
- `Ai_j_nl_base_kin.dat` : Corresponding kinetic energy expression  $[\epsilon - v(r)] \cdot u(r)$
- `Ci_j_nl_base.dat` : `confined` free-atom like radial function  $u(r)$  number  $j$  for `species` number  $i$ , radial and angular quantum numbers  $nl$ .
- `Ci_j_nl_base_pot.dat` : Corresponding basis-defining potential including confining potential
- `El_base_n_l.dat` : Radial function  $u(r)$  after the basis-confining potential was applied for the `species` named  $El$ , radial and angular quantum numbers  $nl$  (same as `Ai_j_nl_base.dat`).
- `El_base_pot.dat` : Basis-defining potential for atomic (minimal) radial functions of the `species` named  $El$ , after addition of the confining potential (as defined by `cut_pot`).
- `El_base_pot.dat` Free-atom density of `species` named  $El$  (same as `El_base_pot.dat`).
- `El_free_n_l.dat`: Radial function  $u(r)$  before the basis-confining potential was applied for the `species` named  $El$ , radial and angular quantum numbers  $nl$
- `El_free_pot.dat` Spherical self-consistent free-atom potential of the `species` named  $El$  (implicitly confined by the `cut_free_atom` potential, but this artificial part is here not included)
- `El_free_rho.dat` Free-atom density of the `species` named  $El$ .
- `Hi_j_nl_base.dat` : `hydro` radial function  $u(r)$  number  $j$  for `species` number  $i$ , radial and angular quantum numbers  $nl$ .

- `Hi_j_nl_base_kin.dat` : Corresponding kinetic energy expression  $[\epsilon - v(r)] \cdot u(r)$
- `Ii_j_nl_base.dat` : `ionic` radial function  $u(r)$  number  $j$  for `species` number  $i$ , radial and angular quantum numbers  $nl$ .
- `Ii_j_nl_base_pot.dat` : Corresponding basis-defining potential including confining potential
- `ty_i_j_n_l.dat` : After on-site orthonormalization, radial function  $u(r)$  of type `ty` (atomic, `confined`, `hydro`, `ionic`, ...), for `species` number  $i$ , radial function number  $j$ , radial and angular quantum numbers  $n, l$ .
- `kin_ty_i_j_n_l.dat` : Corresponding kinetic energy expression after on-site orthonormalization.

**output sub-tag:** `cube` (`control.in`)

Usage: `output cube` type

Purpose: Writes a quantity (density, eigenfunction, ...) to a uniform three-dimensional grid, using the ASCII-based cube file format established by the Gaussian code and accepted by numerous visualization tools.

type is a string, indicating the quantity to be plotted.

The “cube” file format originates from the Gaussian code, but publically available descriptions exist, for example here:

<http://paulbourke.net/dataformats/cube/>

The first two lines serve as comment headers only. The third line contains the number of atoms included, and the origin around which the plot is centered. Lines four, five, and six each contain the number of subdivisions (“voxels”) of the  $x$ ,  $y$ , and  $z$  axis used for the plot, followed by the length vector of each “voxel” (step along the cube direction). The following lines first summarize the underlying atomic structure, before the actual volumetric data appear as a block of formatted ASCII output. For more information and an example, see the description in Sec. 3.22.

A complete specification of a cube output consists of up to 10 lines. Of these, all but one have pre-defined settings and can be omitted.

- `output cube` type This is the only mandatory line, specifying which type of cube file should be produced. FHI-aims presently allows the following options for type:
  1. `delta_density` : Writes the difference between the initial (superposition of free atoms) and the final self-consistent density to a file.
  2. `eigenstate n` : Writes the wave function of the  $n$  – *th* eigenstate *state* to a file. *State* must be an integer number. In periodic calculations, only the real part of the eigenstate is printed. By default, the first spin channel and the first k-point is printed out, see also `cube spinstate` and `cube kpoint`.

3. `eigenstate_density n`. Writes the electron density of the *n*-th eigenstate to a file. The eigenstate density is obtained as the square of the wavefunction. In periodic calculations, this includes the imaginary part, and it might therefore be more useful to look at the eigenstate density rather than at (the real part of) the wave function. By default, the first spin channel and the first k-point is printed out, see also `cube state`.
4. `spin_density`: The spin density  $n^\uparrow(\mathbf{r}) - n^\downarrow(\mathbf{r})$  is written to a file named `spin_density.cube`. Only available for `spin collinear`.
5. `stm`: Must be followed by a real number *V*. Calculates 3D tunneling current map (more precisely, the tunneling current is proportional to the printed values) which can be used to plot STM images for a given voltage *V* (in Volts) in the frame of the Tersoff-Hamann model. This is done by summing up eigenstate densities for all eigenstates between the Fermi level and *V* (in eV), and the result is multiplied by *V*. In addition, a file `cube_xxx_stm_z_map.cube` will be printed. It contains values of the z-coordinate at the vertices of the cube, and can be used along with the tunneling current map to color the constant current isosurfaces according to their extent in the z-direction (to mimic the constant current image contrast in STM imaging). The output of `stm-cubes` works only for periodic systems.
6. `total_density`: The full electron density is recomputed and written to the a. In case of a periodic calculation, electron density from all unit cells that overlap with the cube output region will be printed.
7. `long_range_potential`: Prints the long range electrostatic potential of the Ewald summation. This result is useful in regions where no electron density is found and is much faster than the output of the full potential.
8. `potential`: The whole (i.e, short-range and long-range) electrostatic potential is recomputed on a cube grid and written out. Presently, this is an experimental feature, and the results should be carefully checked.

Units for densities and eigenstates are  $\text{\AA}^{-3}$  and  $\text{\AA}^{-3/2}$ , respectively.

- `cube spinstate spin` This keyword allows to choose whether to print spin-channel 1 or 2. Default:1 **Attention: The deprecated keyword `cube spin` DOES NOT change the spin-channel. `cube spin` is a synonym for `cube spinmask` (see below)**
- `cube kpoint kpoint` This keyword allows to choose the k-point to be printed. *kpoint* is an integer number following the same ordering as the k-points within the SCF-cycle. It is presently not possible to output cube files at a k-point not included in the scf. Default: 1.
- `cube state spin k-point`. Deprecated keyword to choose spin and k-point. If omitted, this keyword defaults to `cube state 1 1`. Note that for cluster calculations, *k-point* must always be 1.
- `cube filename name_of_the_file` Allows to customize the name of the cubefile. If this line is not given, FHI-AIMS will default to a file name which contains the

number of the cube requested, its type, and, if applicable, the corresponding spin and k-point of the data.

- `cube format format` Apart from the default cube format, FHI-AIMS also supports output in the formats of the gOpenMol and XCrysden software. This is requested by the line `cube format format`. The options for *format* are `cube`, `gOpenMol`, and `xsf` (the XCrysden format). Default: `cube`
- `cube divisor number` This is a technical settings which governs the parallelization of the cube output. The whole cube is divided into smaller, so-called minicubes, which are then treated one after the other. The value governs the number of points in each directed to be used for each minicube, i.e., its size. A larger setting therefore results in less minicubes (and it thus potentially faster), but also a larger demand for memory. Unless there are problems with memory, this setting usually does not need to be touched, with the exception of output potential, where it should be set to its maximal value (45), independent of the number of processors used. Default: 10
- `cube spinmask` For `total_density` cube files, the line `cube spinmask i j` with integer *i* and *j* allows to manipulate the spin channels independently according to the following formula:  $i \cdot n^\uparrow(\mathbf{r}) - j \cdot n^\downarrow(\mathbf{r})$ . If *i* = 1 and *j* = 1 (which is the default), the total density will be computed as normal. This keyword replaces the earlier keyword `cube spin`
- `cube origin xyz` Single line which specifies the origin, i.e., the center of the region to be plotted. Values are given in Å. If omitted, the same origin as for the previous cube file is used. If no origin has been given yet, it defaults to the geometric center of the molecule in the cluster case or (0,0,0) for periodic calculations. For slab type calculations (when using `dipole_correction .true.`), the origin is set to the center of the slab.
- `cube edge n dx dy dz` Specifies the edges of the volumetric data to be plotted. Separate lines have to be given for each of the three edges of the cube. In each line, *n* indicates the number of steps a particular edge (voxel), and *dx*, *dy*, *dz* indicate the length of each individual step [i.e., the full cube edge length is ( $n \cdot dx$ ,  $n \cdot dy$ ,  $n \cdot dz$ )]. If omitted the same edges as for the previous cube file are used. If no edges have been specified yet, FHI-AIMS defaults in the cluster case to orthogonal grids of 0.1Å length, which span the whole molecule plus 14 Bohr beyond the outermost nuclei. For periodic calculations, the default edges are the same as the lattice vectors, again with 0.1Å step length. *Note: In some cases, such as separated molecules or surfaces with large amounts of vacuum, the defaults might be far from ideal and result in excessively large files. Presently, the code will stop if the defaults imply cube files with more than 650MB. Users are always strongly encouraged to specify cube geometries themselves.*

Example for a spin-polarized non-periodic system:

```
output cube total_density
```

```

cube origin 1.59 9.85 12.80
cube edge 101 0.15 0.0 0.0
cube edge 101 0.0 0.15 0.0
cube edge 101 0.0 0.0 0.15
output cube eigenstate 151
cube spinstate 1
output cube eigenstate 151
cube spinstate 2
output cube eigenstate 152
cube spinstate 1
output cube eigenstate 152
cube spinstate 2
output cube eigenstate 153
cube spinstate 1
output cube eigenstate 153
cube spinstate 2
output cube eigenstate 154
cube spinstate 1
output cube eigenstate 154
cube spinstate 2

```

So this would first request cube output for the total density. See the details above on how to get individual versions of the spin density. Then, the **center** of the cube is specified at (1.59, 9.5, 12.80) Å. Each cube direction has 101 points that are spaced apart by 0.15 Å, giving a total cube edge length of 15 Å in each direction. Finally, output is requested for the Kohn-Sham wave functions associated with eigenstates number 151-154. For each eigenstate, the additional cube state  $i$  line requests first the spin-up channel ( $i=1$ ), then spin-down ( $i=2$ ).

**NOTE** that for periodic systems, the cube state lines would have to give two numbers, one for the spin state and one for the  $k$  point. By default, only  $k$ -point number 1 is printed. For unshifted  $k$  grids, that is the  $\Gamma$  point.

---

**output sub-tag:** `density` (control.in)

Usage: `output density`

Purpose: Writes the electron density  $n(\mathbf{r})$  at each integration grid point  $\mathbf{r}$  to a file `density.dat`.

Note that this density output is *not* given on a uniform grid, but simply on the overlapping atom-centered grid used for all internal operations of FHI-aims. For a density on a uniform grid, see the `output cube` subkeyword.

`output density` additionally writes the difference between the current density and the superposition-of-free-atom reference density to a file `diff-density.dat`

---

**output sub-tag:** `dipole` (control.in)

Usage: `output dipole`

Purpose: Calculates and writes the electrical dipole moment of the structure to the FHI-aims standard output as a post-processing step.

This is generally useful, but the dipole moment is particularly needed to compute molecular oscillator strengths for individual vibrational frequencies.

Note that, for charged systems, the electrical dipole is defined with respect to the (possibly arbitrary) origin of the file `geometry.in`, rather than an origin within the system itself. Thus, charged systems will yield different dipole moments for different choices of origin, but the important dipole *differences* needed to compute, e.g., oscillator strengths via finite differences remain well-defined.

**output sub-tag: dos** (`control.in`)

Usage: `output dos` Estart Eend n\_points broadening

Purpose: Writes the density of states (DOS) to an external file for plotting purposes.

Estart : Lower bound of the single-particle energy range for which the DOS is given.

Eend : Upper bound of the single-particle energy range for which the DOS is given.

n\_points : Number of energy data points for which the DOS is given.

broadening : Gaussian broadening applied to obtain a smooth density of states based on the peaks produced by individual states.

This keyword shares its syntax with `output atom_proj_dos` and `output species_proj_dos`. See also section 4.4 for more details.

Two output files emerge from this option:

- `KS_DOS_total_raw.dat` contains the total DOS components as a function of the eigenvalue energy (first column) as used internally in FHI-aims. The energy zero is then given by the vacuum level (non-periodic systems) or by the  $G=0$  component of the long-range Hartree potential (periodic systems).
- `KS_DOS_total.dat` contains the same information, except that the energy zero is shifted to the Fermi energy (metallic systems) or valence band maximum (insulators), respectively.

When followed by the following keyword

`dos_kgrid_factors`  $n_1$   $n_2$   $n_3$

where  $n_1$ ,  $n_2$  and  $n_3$  are integers, the dimensions of the k-point grid along the first, second and third lattice vectors are multiplied by  $n_1$ ,  $n_2$  and  $n_3$ , respectively. New eigenvalues are re-calculated non-selfconsistently on the new denser k-point grid. The new eigenvalues are then used to plot improved (so-called perturbative) density of states. By default, the original k-point grid is used.

---

**output sub-tag: eigenvectors** (control.in)

Usage: `output eigenvectors`

Purpose: Writes the actual Kohn-Sham eigenvectors  $c_{il}$  into separate files for each spin channel.

Restriction: This functionality is best tested for periodic systems with `KS_method lapack` at present. For periodic systems, it has no effect unless specific band structure output is requested through `output band`.

For *non-periodic* systems, this option causes the Kohn-Sham eigenvectors  $c_{il}$  (for basis functions  $i$ , eigenstates  $l$ ) to be written out for each state, whenever the Kohn-Sham eigenvalues are written. However, at present the non-periodic version is mostly of 'debug' character. In particular, `KS_method scalapack` is not supported under all circumstances. Please test carefully.

For *periodic* systems, eigenvector output must be requested together with the `output band` functionality described above. If `output eigenvectors` is set in addition, the Kohn-Sham eigenvectors  $c_{il}(\mathbf{k})$  will be written into separate files for each spin channel, *only* for each  $k$ -point for which band output was requested.

For each state  $l$ , the real and imaginary part of  $c_{il}(\mathbf{k})$  will then be written out for each basis function  $i$ . Output file names are assigned automatically based on the number of the output band, and to the specific  $k$ -point in that band, to which they pertain, e.g.:

```
KS_eigenvectors.band_*.kpt_*.out
```

In addition to the eigenvectors themselves, these files also contain as header information:

- the *relative* coordinates of the  $k$ -point in question (in units of the reciprocal lattice vectors of the structure in question)
- Information on the identity (atom number and angular momentum) for each basis function
- The Kohn-Sham eigenvalue and occupation number of each state.

The listed eigenvectors pertain to the superimposed, Bloch-like basis functions (with phase factors!)  $\chi_{i,k}(\mathbf{r})$  as defined through Eq. (22) of the FHI-aims Computer Physics Communications description, Ref. [12].

---

**output sub-tag: elpa\_timings** (control.in)

Usage: `output elpa_timings`

Purpose: Writes timings for the parallel Elpa eigenvalue solver to standard output.

---

**output sub-tag: grids** (control.in)

Usage: `output grids`

Purpose: Writes to separate files: (i) the `radial_base` integration grid shells for each species incl. integration weights, and (ii) the full three-dimensional grid point locations incl. integration weights.

**output sub-tag:** `hessian` (`control.in`)

Usage: `output hessian`

Purpose: Writes the initial Hessian approximation for the structure optimizer into the file `hessian.out`.

**output sub-tag:** `hirshfeld` (`control.in`)

Usage: `output hirshfeld`

Purpose: Produces a Hirshfeld analysis of partial charges and moments on each atom.

Restriction: Currently disabled for Hartree-Fock and hybrid functionals as the underlying spherical free-atom charge density is not produced on a dense 1-dimensional radial grid for these functionals. Support for this can easily be added by commenting out the “stop” line in the source code, except that the Hirshfeld analysis is then based on the DFT-LDA free atom density  $n_{\text{at}}^{\text{free}}(r)$ .

Defining “atoms-in-molecules” is a classic, intuition based problem; one would like to associate individual (partial) charges or moments with individual atoms in a bonded structure. This process is by necessity not uniquely definable (molecules *are* not atoms, and there are no rigorously defined boundaries between atoms). Nonetheless, much chemical intuition is based on such a concept.

Hirshfeld’s [50] “atoms-in-molecules” partitioning relies on the same idea as the partitioning of our charge density for the electrostatic potential (Eq. 3.17), using the free-atom electron density  $n_{\text{at}}^{\text{free}}(r)$  as the weight function  $g_{\text{at}}(\mathbf{r})$  in Eq. (3.13). Since (for a given functional), we know the spherical  $n_{\text{at}}^{\text{free}}(r)$  exactly, this analysis remains well-defined even as external circumstances such as the basis set change. However, the resulting values are still qualitative in the sense that Hirshfeld’s [50] “atoms-in-molecules” partitioning is just one among many other prescriptions that have been suggested in the literature.

**output sub-tag:** `k_eigenvalue` (`control.in`)

Usage: `output k_eigenvalue` number

Purpose: For periodic structures, determines for how many  $k$  points FHI-aims will write the electronic eigenvalues  $\epsilon_l(\mathbf{k})$ .

number is an integer number. Default: 1.

Eigenvalues will only be written for the first number  $k$ -points by default. For dense  $k$ -grids, the sheer number of  $k$ -points simply gets too large to allow for a full output.

---

**output sub-tag:** `k_point_list` (control.in)

Usage: `output k_point_list`

Purpose: For periodic geometries only, this option writes a complete listing of the reciprocal space coordinates of all  $k$ -points in the calculation to the standard output file.

The  $k$ -point coordinates are written in units of the reciprocal lattice vectors of the structure. This option is also the default when using `output_level` full .

---

**output sub-tag:** `matrices` (control.in)

Usage: `output matrices`

Purpose: Writes the Hamiltonian and overlap matrices  $s_{ij}$  and  $h_{ij}$  into separate files.

Restriction: This functionality is unavailable for periodic systems, or if a `packed_matrix_format` is used.

---

**output sub-tag:** `matrices_parallel` (control.in)

Usage: `output matrices_parallel types [format]`

Purpose: Writes Scalapack/Elpa matrices into separate files.

`types` is a string that determines the type of matrices that are written.

`format` (optional) is a string that determines the format of the output.

Depending on option “types”, the upper part of up to three different matrices is written into separate files. If `types` is “n” (without quotes), no matrices are written. If `types` is set to “h”, then the Hamiltonian is written. The choice “o” causes the overlap matrix to be output. Finally, “s” refers to the system matrix from which the eigenvalues are calculated. The last three options can be combined. For example, “ho” means Hamiltonian and overlap.

The format of the files can be controlled with the optional parameter `format`. Possible values are “asc” for ASCII output (default) and “bin” for binary output.

---

**output sub-tag:** `mulliken` (control.in)

Usage: `output mulliken`

Purpose: Produces a Mulliken analysis of the occupation of each atom and its angular momentum channels in terms of the basis used.

Restriction: This option is available as a per-atom summary only if ScaLapack is used.

Defining “atoms-in-molecules” is a classic, intuition based problem; one would like to associate individual (partial) charges or moments with individual atoms in a bonded structure. This process is by necessity not uniquely definable (molecules *are* not atoms, and there are no rigorously defined boundaries between atoms). Nonetheless, much chemical intuition is based on such a concept.

A classic “atoms-in-molecules” concept is the Mulliken analysis [79], which defines electronic occupations of atoms by projected occupations into the localized basis functions associated with them (see the standard literature for exact definitions and use).

In short, when so requested, FHI-aims will provide a decomposition of the electronic density per atom, angular momentum channel, and possibly spin channel, allowing to deduce approximate partial charges. The summarized Mulliken analysis is written into the standard output stream, while a separate file `Mulliken.out` contains detailed state-by-state projected electron occupations.

Note that a Mulliken analysis is somewhat ill-defined because of basis function overlap; thus electrons can be counted to one atom or another at will. For small basis sets, a Mulliken analysis may still yield qualitatively reasonable numbers, but it becomes increasingly ill-defined as the atom-centered basis sets approach basis set completeness.

---

**output sub-tag:** `nuclear_potential_matrix` (`control.in`)

Usage: `output nuclear_potential_matrix`

Purpose: Writes the matrix elements of only the bare electron-nuclear potential in the current basis set to a file

Restriction: This functionality is unavailable for periodic systems, or if a `packed_matrix_format` is used.

This can be useful if the FHI-aims basis functions are needed for a further, separate purpose (Quantum Monte Carlo etc) but please note that the integration accuracy for the Coulomb singularity near the nuclei must be higher than in our standard calculations (where the singularity is cancelled by the kinetic energy), so increasing `radial_multiplier` is in order.

---

**output sub-tag:** `ovlp_spectrum` (`control.in`)

Usage: `output ovlp_spectrum`

Purpose: Writes the non-singular part of the eigenvalue spectrum of the overlap matrix to the FHI-aims standard output.

Restriction: Works only for the cluster case, and only for `KS_method lapack`.

This option can help show if (or if not) the chosen basis set for the full system is close to ill-conditioning (see the comments for keyword `basis_threshold`).

**output sub-tag: quadrupole** (`control.in`)

Usage: `output quadrupole`

Purpose: Calculates and writes the electrical quadrupole moment of the structure to the FHI-aims standard output as a post-processing step.

**output sub-tag: rho\_multipole** (`control.in`)

Usage: `output rho_multipole`

Purpose: Writes the partitioned atom-centered charge multipole components  $\delta\tilde{n}_{\text{at},lm}(r)$  to individual files for each atom,  $l$ , and  $m$  (see Eq. 3.17).

**output sub-tag: species\_proj\_dos** (`control.in`)

Usage: `output species_proj_dos Estart Eend n_points broadening`

Purpose: Writes an projected, angular-momentum resolved partial density of states (pDOS) averaged over all atoms of each `species`.

`Estart` : Lower bound of the single-particle energy range for which the pDOS are given.

`Eend` : Upper bound of the single-particle energy range for which the pDOS are given.

`n_points` : Number of energy data points for which the pDOS are given.

`broadening` : Gaussian broadening applied to obtain a smooth partial density of states based on the peaks produced by individual states.

This option is based on a Mulliken Analysis and shares its syntax with `output dos` and `output atom_proj_dos`. See also section 4.4 for more details.

Different from the `atom_proj_dos` option, this option writes its output averaged over all atoms of each `species` defined in `control.in` and used in `geometry.in`. This provides a quick handle to obtain averaged pDOS's for well-defined subgroups of individual atoms, e.g., those of a given layer of a slab, by simply defining a separate `species` for the desired atoms in `control.in`.

There are two types of output files for each atom:

- `species_l_proj_dos_raw.dat`, where `species` denotes the `species` name used in `geometry.in` and `control.in`. This file contains the total and angular-momentum resolved DOS components as a function of eigenvalue energy (first column) as used internally in FHI-aims. The energy zero is then given by the vacuum level (non-periodic systems) or by the  $G=0$  component of the long-range Hartree potential (periodic systems).
- `species_l_proj_dos.dat`, which gives the same information, except that the energy zero is shifted to the Fermi energy (metallic systems) or valence band maximum (insulators), respectively.

Note that projected densities of states such as given here must be based on some kind of projection orbitals, the choice of which is somewhat arbitrary by necessity. This is thus a tool for *qualitative* analyses.

In FHI-aims, we project directly on the atom-centered angular-momentum components as defined by the *overlapping* basis set. This definition becomes the more arbitrary the larger the basis set, just like a `mulliken` analysis. The closer the full basis comes to completeness, the more ambiguous will a `mulliken`-like analysis become, since it may not be *a priori* clear which electrons should be counted towards the basis functions of one atom vs. those of another atom. Thus, do *not* expect a pDOS to simply converge as the basis set size is increased; use it as a qualitative indicator of trends, but nothing more.

**output sub-tag:** `v_eff` (`control.in`)

Usage: `output v_eff`

Purpose: Writes the local effective potential  $v_{\text{eff}}(\mathbf{r})$  at each integration grid point  $\mathbf{r}$  to a file `v_eff.dat`.

Note that the meaning of this effective potential depends on the `xc` option used. For DFT-LDA, this is simply the full local potential. For gradient-corrected (GGA) functionals, the gradient partial derivatives of the exchange-correlation functional are *not* included in the potential, as they are treated separately by integration by parts (see Ref. [12]). For hybrid functionals or Hartree-Fock, the exchange part of the potential is of course not included.

Note also that this output does *not* happen on a uniform grid. For further processing, a proper visualization tool is needed, and/or an interpolation onto a uniform grid must be done.

**output sub-tag:** `v_hartree` (`control.in`)

Usage: `output v_hartree`

Purpose: Writes the electrostatic (Hartree) potential multipole components  $\delta\tilde{v}_{\text{at},lm}(\mathbf{r})$  to individual files for each atom,  $l$ , and  $m$ .

---

**output sub-tag:** `zero_multipoles` (`control.in`)

Usage: `output zero_multipoles`

Purpose: Prints out the partial charges associated with the multipole electrostatic potential of each atoms in each s.c.f. iteration.

The resulting values are “atoms-in-molecules” like partial charges assigned to each atom, similar to a `hirshfeld` partitioning (but not identical because a different partitioning function may be used as the `hartree_partition_type`).

### **3.23 Deprecated keywords**

The following section lists a number of keywords in FHI-aims which exist, but which may go away in future versions of FHI-aims. In some cases, this is because the relevant modifications proved successful, and there is no sense in maintaining some old, obsolete extra functionality without any use in production settings. In other cases, the relevant keywords were experiments that did not yield the anticipated success, and / or functionality that may be superseded in a different, more comprehensive way in the future.

---

## Tags for general section of `control.in`:

---

### Tag: `Adams_Moulton_integrator` (`control.in`)

Usage: `Adams_Moulton_integrator` flag

Purpose: Allows to switch between a simple integrator and the higher-order Adams-Moulton integration scheme to determine the Hartree potential components from classical electrostatics.

flag is a logical string, either `.false.` or `.true.` Default: `.true.`

---

### Tag: `batch_distribution_method` (`control.in`)

Usage: `batch_distribution_method` method

Purpose: Parallel distribution of integration grid batches *only* in the case that the external qhull and METIS libraries are configured.

method is a string, the only possible value being `qhull+metis` at this point.

Outsources the distribution of integration grid batches to the external qhull and METIS libraries. Only relevant if these libraries are compiled into the code. However, the associated `grid_partitioning_methods` are less useful than the default `maxmin` algorithm, and the internal work distribution method of FHI-aims usually performs rather well. Therefore, this option is deprecated and kept only for experimental purposes, for now.

---

### Tag: `communication_type` (`control.in`)

Usage: `communication_type` type

Purpose: Determines the type of calculation / storage of per-atom spline arrays of the Hartree potential for a parallel run.

type is a string, see below. Default: `calc_hartree` .

In a parallel run of FHI-aims, each processor holds a certain part of the real-space integration grid, which in turn are each touched by all atom-centered multipole components (splined) of the real-space Hartree potential. So, to construct the electrostatic (Hartree) potential on each grid point, an array of splined atom-centered multipole components  $\delta\tilde{v}_{\text{at},lm}(r)$  must be available on every MPI sub-process (see Sec. 3.7 for details regarding the electrostatic potential). The memory use to store these components grows rapidly and with a large prefactor with system size. Thus, keeping a local copy of all the splined multipole components of the Hartree potential on each CPU is not advisable.

The actual handling of these components is instead controlled by keyword `communication_type`.

The following choices for option are possible:

`calc_hartree` : The default, and usually very efficient. Hartree potential components for each atom are integrated on the fly on each CPU when needed (usually less CPU time than communication time).

`shmem` : If compiled with shared memory support (see Section 1.4), each compute node of a parallel run keeps the components of the Hartree potential in a separate shared memory segment, only internode communication is needed. Performance test show hardly any benefit over `calc_hartree`. Note that the (legacy) keyword `distributed_spline_storage` should be false for `shmem`, at least.

---

**Tag:** `force_new_functional` (control.in)

Usage: `force_new_functional` flag

Purpose: For test purposes, allows to switch to an energy functional form that treats the electronic and nuclear electrostatic energy terms separately.

flag is a logical string, either `.false.` or `.true.` Default: `.true.`

See Ref. [12] regarding the correct shape of the energy functional that treats the nuclear and electronic parts of the electrostatic energy together on a per-atom basis.

---

**Tag:** `force_smooth_cutoff` (control.in)

Usage: `force_smooth_cutoff` tolerance

Purpose: Optionally, enforces smoothness of all basis functions near the cutoff radius.

tolerance is a small positive real number. Default: No check.

If requested, keyword `force_smooth_cutoff` ensures that the radial function  $u(r)$  and its first and second derivatives remain below tolerance at the outermost point of the logarithmic grid where any of them is non-zero at all. The code stops if the onset of the radial function is too abrupt.

It would be a good idea to switch this option on if reducing the width parameter of keyword `cut_pot` to a very low value (say, below 1 Å).

---

**Tag:** `grouping_factor` (control.in)

Usage: `grouping_factor` factor

Purpose: Grouping factor for the (experimental, and not recommended!) group `grid_partitioning_method`.

factor is an integer number, describing how close-by grid points are grouped together. Default: 2.

This keyword is retained for experimental purposes only, for the moment. Since the related `grid_partitioning_method` group was a proof-of-concept to show that the default `maxmin` performs better, this keyword is now deprecated. See Ref. [44] for more details if interested.

---

**Tag:** `hartree_worksize` (control.in)

Usage: `hartree_worksize` megabytes

Purpose: Limits the size of workspace arrays used to construct the Hartree potential on each CPU.

megabytes : The maximum allowed work space size to construct the Hartree potential, in megabytes. Default: 200 MB.

Several large work space arrays across the integration grid are used in the construction of the Hartree potential. Their size can grow quite large, especially when forces are computed for large structures (then, three arrays per atom are required for all atoms).

FHI-aims can circumvent this by computing the final output (integrated energies and forces) in “chunks” of the whole integration grid, limiting the work space used for each chunk. This modification is especially important on low-memory-per-processor architectures such as the BlueGene.

---

**Tag:** `KH_post_correction` (`control.in`)

Usage: `KH_post_correction` flag

Purpose: *Under construction – do not use* A way to replace the scaled ZORA post-processing correction for scalar relativity by a Koelling-Harmon type scalar-relativistic correction.

flag is a logical string, either `.false.` or `.true.` Default: `.false.`

This keyword is no longer supported, do not use it. It will be superseded by an improved handling of scalar relativity during the s.c.f. cycle in the future.

---

**Tag:** `mixer_swap_boundary` (`control.in`)

Usage: `mixer_swap_boundary` bytes

Purpose: Ignored; never swap. Used to allow to swap the stored density components for Pulay mixing to disk if they exceed a certain memory boundary.

On few-CPU systems and for mid-sized systems (several hundred atoms), the stored electron density components from past iterations are a large part of the memory used. If this becomes a bottleneck, the stored Pulay arrays can in principle be swapped to disk, instead, to be read only during Pulay mixing.

*If anyone has a strong need for this currently unsupported feature, please contact us.*

---

**Tag:** `multiplicity` (`control.in`)

Usage: `multiplicity` value

Purpose: If set, specifies the multiplicity of the system.

Restriction: Currently available for non-periodic geometries only. Use `fixed_spin_moment` instead.

value : integer number, sets the overall multiplicity as  $2S + 1$ .

Meaningful only in the spin-polarized case (`spin collinear` in `control.in`). On a technical level, this is a special case of the more general, locally spin-constrained DFT formalism available within FHI-aims (see Sec. 3.14). Note that the underlying `constraint_electrons` keyword can be used to enforce a *non-integer* fixed spin moment, in addition to allowing to fix electron or spin numbers in given subsystems

**Tag:** `occupation_thr` (`control.in`)

Usage: `occupation_thr` value

Purpose: Any occupation numbers below value will be treated as zero. value is a small positive real number. Default: 0.d0 .

**Tag:** `recompute_batches_in_relaxation` (`control.in`)

Usage: `recompute_batches_in_relaxation` flag

Purpose: Allows to switch off the redistribution of atom-centered grid points into new integration batches after a relaxaton step.

flag is a logical string, either `.false.` or `.true.` Default: `.true.`

For practical purposes, the integration grid should always be repartitioned after a relaxation step; the associated overhead is low, and the shape of the batches will remain optimal in the face of individual points that “move” with different atoms.

**Tag:** `squeeze_memory` (`control.in`)

Usage: `squeeze_memory` flag

Purpose: Used to allow one combined workspace for three different purposes.

This option is no longer necessary due to optimizations by Rainer Johanni. flag is a logical string, either `.false.` or `.true.` Default: `.false.`

**Tag:** `use_angular_division` (`control.in`)

Usage: `use_angular_division` flag

Purpose: If radial grid shells are used as integration batches, allows to switch off their subdivision into “octant” batches.

flag is a logical string, either `.false.` or `.true.` Default: `.true.`

---

This flag currently only applies to the initialization iteration, in case that self-adapting angular grids are used (not recommended; see keyword [angular\\_acc](#) if interested). Even then, switching off the subdivision of radial shells can only decrease the performance.

## Subtags for *species* tag in `control.in`:

---

**species sub-tag:** `cut_core` (`control.in`)

Usage: `cut_core` type [radius]

Purpose: Can be used to define a separate (tighter) onset of the cutoff potential for all `core` radial functions.

`type` : A string, either `finite` or `infinite`. Default: `finite` .

`radius` : A real number, in Å: Onset radius for the cutoff potential, as defined in the `cut_pot` tag. Default: same as `onset` in `cut_pot`.

Deprecated flag because `basis_dep_cutoff` should supersede this functionality in a more organized way. Having a separate, tighter cutoff for core radial functions sounds like a good idea, but core radial functions are already rather localized anyway. Our experience is that the separate core setting either does not matter for CPU time, or already introduces noticeable total energy changes when it matters.

## Chapter 4

# Running FHI-aims: Guides to specific tasks

The complete specification of all keywords available in FHI-aims and their capabilities was given in the previous chapter. The present chapter attempts to illustrate these capabilities in a tutorial way. The purpose of this exercise is threefold: (i) to provide clear starting points for the most common types of production calculations; (ii) to provide clear examples that can be modified (but need not be completely reinvented) for the many further capabilities of FHI-aims; (iii) to describe specific “higher-level” tasks that bind together multiple FHI-aims calculations by way of scripts (e.g., vibrations, or nudged-elastic band calculations to find transition barriers).

## 4.1 Ground state DFT: Total energies and relaxation

As a first and basic example, we discuss how to set up a simple DFT total-energy calculation for a given structure in FHI-aims. We here expand on the example provided as a *testrun*: The relaxation of a H<sub>2</sub>O molecule towards its minimum-energy structure. The relevant input files `geometry.in` and `control.in` are included in the directory *testrun/H2O-relaxation* (see Chapter 1).

The starting geometry here is badly distorted H<sub>2</sub>O, with an initial bond angle of 90°. For any relaxation runs, we strongly recommend a two-step procedure: First, pre-relax the structure with *light* settings down to (say) 10<sup>-2</sup> eV/Å or so, and only then follow up with a post-relaxation run using *tight* settings or anything else. *light* calculations may easily be cheaper by a factor of 5-10 than *tight* ones, and going down a relaxation trajectory of any length can then be a tremendous waste of computational resources.

The test case below only includes the quick but safe prerelaxation step, leading to an improved geometry that is the optimum using *light* settings. Based on this resulting geometry, the postrelaxation step with *tight* settings should be simple follow-up exercise.

### Input files

Turning first to `geometry.in`, we see that the basic geometry input for FHI-aims is very simple in most cases: `atom` lines that contain nuclear coordinates in Å, together with the appropriate `species` designation (in this case, H and O). For a spin-polarized calculation, one might additionally want to specify initial spin moments for selected atoms using the `initial_moment` keyword, in order to define a good initial spin density guess for the s.c.f. procedure.

The input file `control.in` contains all necessary computational information regarding the desired run. Most importantly, the `xc` keyword is required to specify the exchange-correlation functional; FHI-aims will not proceed without this information. The further “physical” specifications – `spin`, `relativistic`, and `charge` – are all at their default settings (no spin-polarization, no relativity, and no charge), but are listed explicitly to make them visible at a quick glance. This is especially important for the `relativistic` keyword, where the `none` setting would not be justified for heavier elements (see Sec. 4.2).

The next few keyword concern the technicalities of obtaining self-consistency: a low `occupation_type` broadening by 0.01 eV of occupation numbers around the Fermi level (this has no physical impact in a molecule with a HOMO-LUMO gap), specifications for the pulay `mixer`, and convergence criteria for the s.c.f. cycle. The criteria referring to energies and the density are here set tightly, in order to have forces that are already mostly converged when the (more expensive) force computation is first done.

The last general setting, `relax_geometry`, specifies a geometry relaxation using the `bfgs` algorithm, together with a standard convergence criterion for the forces in the final geometry: No force component for any atom of the relaxed structure should exceed 10<sup>-2</sup> eV/Å. This criterion may well be set tighter for sensitive cases, such as a starting

geometry to obtain vibrational frequencies, but not orders of magnitude tighter (for example, a setting of  $10^{-4}$  eV/Å might end up just probing some residual numerical noise, for example from the finite integration grids, with no noticeable geometry or total energy changes resulting at all).

The version of the BFGS optimization algorithm used here is a trust-radius enhanced version (as compared to a straight, textbook-like BFGS implementation which could alternatively be used, see the description of the `relax_geometry` keyword). By default, the convergence of the relaxation is additionally sped up by an intelligent guess for the Hessian matrix used in the initial BFGS step. This is done by way of a slightly modified version of the general purpose model matrix proposed by Lindh and coworkers [70], see keyword `init_hess`.

In `control.in`, it remains to set the `species` information for H and O, the elements included in `control.in`. Normally, these settings should be obtained by copy-pasting the relevant information from the `species_defaults` directory, for example the choices in the `light` or `tight` subdirectory located there. Once this is done, the `species` defaults may still be adapted for the purpose in question.

## Output stream

We next analyze some significant parts of the standard output produced by FHI-aims, also provided in the file `H2O.reference.out`. We emphasize that this output is kept as human-readable as possible; it pays to actually *look* into the output, especially when something does not appear to have gone correct. Often, a simple warning in the initial input section or elsewhere in the file may already tell you what is going on. Warnings can also be identified by “grepping” for asterisks in the file.

The standard output stream is structured as follows:

- A summary of the setup (nodes used, required fixed dimensions, information in `control.in` and `geometry.in`) and, importantly, default values inserted for parameters that were *not* explicitly specified in `control.in`.
- Preparation of fixed parts of the calculation – most importantly, information regarding the setup of the per-species basis
- Initialization – information on the setup of all three-dimensional integrations, and solution of the first Kohn-Sham eigenvalue problem for the initial superposition-of-free-atoms electron density.
- Process and total energy information for each successive s.c.f. iteration.
- Upon convergence of the s.c.f. cycle, the Kohn-Sham eigenvalues are also included, as well as final total energies and forces in a long format for reading by external utilities (scripts etc.)
- This is followed by information on the geometry optimization, up to the coordinates predicted for the next step, based on the converged forces obtained from the previous iteration.



```

atom      0.00000000    -0.07326360    0.00000000  O
atom      0.76740642    -0.67036820    0.00000000  H
atom     -0.76740642    -0.67036820    0.00000000  H

```

---

*Timing information:*

Finally, we note that FHI-aims also provides detailed timing information for each s.c.f. iteration, and as a summary at the end of each run. For example, the provided test run reads similar to this:

---

```

Leaving FHI-aims.
Date      : 20110113, Time      : 160442.031

Computational steps:
| Number of self-consistency cycles      :          39
| Number of relaxation steps              :           4

Detailed time accounting      : max(cpu_time)   wall_clock(cpu1)
| Total time                    :          4.028 s       10.275 s
| Preparation time               :           0.067 s        3.245 s
| Grid partitioning              :           0.097 s        0.214 s
| Preloading free-atom quantities on grid :           0.046 s        0.108 s
| Free-atom superposition energy    :           0.022 s        0.048 s
| Total time for integrations       :           1.412 s        1.946 s
| Total time for solution of K.-S. equations :           0.147 s        0.180 s
| Total time for EV reorthonormalization :           0.007 s        0.011 s
| Total time for density & force components :           1.563 s        2.216 s
| Total time for mixing            :           0.068 s        0.265 s
| Total time for Hartree multipole update :           0.142 s        0.174 s
| Total time for Hartree multipole sum  :           0.558 s        1.576 s
| Total time for total energy evaluation :           0.013 s        0.008 s

```

Have a nice day.

---

The date and time at the end are in the `ddmmyyyy` and `hhmmss.mmm` formats of a wall-clock time, *not* in seconds; i.e., the above calculation did *not* take 222254 s, but rather *ended* at 16:04:42 h, one fine January 13.

In addition, detailed timing is provided both for as elapsed CPU time (on individual CPUs during the run), and actual elapsed wall clock time. One interesting detail to note is that, in the present example, there is a large difference between the reported CPU times and the wall clock times. In a normal production run, such significant discrepancies should *not* occur, and could indicate serious problems with load balancing or communication in a parallel run. For example, the run quoted above was run from within a virtual machine on a laptop computer whose host operating system had other things to do. As a result, a large amount of real (wall clock) time was spent not doing any computations. On a dedicated production machine, this should not happen.

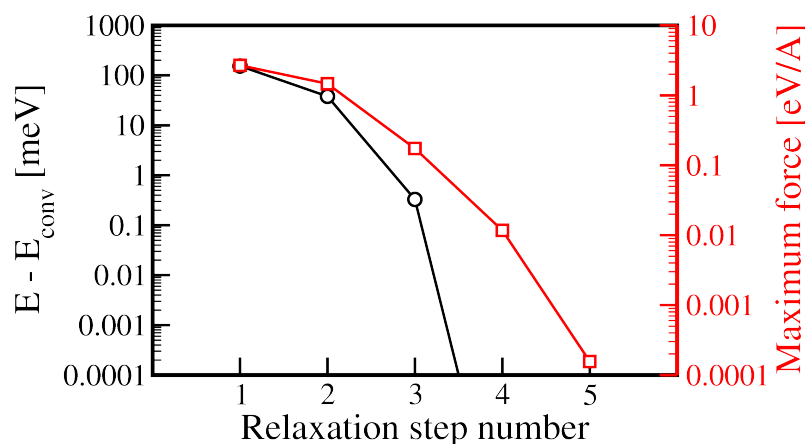


Figure 4.1: Total energy convergence (left  $y$  axis) and force convergence (right  $y$  axis) during the relaxation test run of  $\text{H}_2\text{O}$ . A total of five steps is taken (initial geometry plus four relaxation steps). The total energy of the final step is taken as the reference here, and the total energy in the second-to-last step is already identical within the numerical accuracy of the calculation. The final step is only taken to bring the residual forces (total energy gradients) to practically zero as well.

## Further analysis

Some statistics on the complete relaxation run can be obtained using the script `get_relaxation_info.pl`, located in the `utilities` subdirectory of the FHI-aims distribution. This script is invoked as

```
> get_relaxation_info.pl < H2O.reference.out > statistics.dat
```

or with any other output file. The script searches the FHI-aims output file for total energies (no entropy correction) and maximum force components at the end of each relaxation step. The development of the total energy, total energy difference with respect to the starting geometry, and maximum force component can then be visualized as a function of the relaxation step number, using standard tools such as `xmgrace`. Fig. 4.1 visualizes the progress of the relaxation run based on the data obtained from `get_relaxation_info.pl`.

Likewise, much other information can and should be extracted from the standard output using similar scripts. For example, the development of the geometry can be visualized in the format of a `.xyz` file using the `create_xyz_movie.pl` script, using standard visualization tools such as `jmol` or `vmd`.

## Next steps

As with any electronic structure code, **monitoring the convergence of the basis set for each element is an important user task**, to make sure any physical conclusions are accurate. For instance, the present example could be continued as follows:

- A “prerelaxation” such as the present test run should not employ a huge basis set, simply for efficiency’s sake. The *light* settings used for the present elements use a *tier 1* basis set. Very often, geometries obtained with *light* are already rather close to converged.
- For these same elements, you will find that the *tight* settings actually employ *tier 2*, which is significantly larger and very close to basis set convergence at least for DFT methods.
- Obviously, one might extend this to *tier 3* or higher for test purposes, possibly even based on *really\_tight* settings otherwise. Comparing the changes made between *light*, *tight*, and *really\_tight* settings, and different basis sets, is an interesting exercise. For the H<sub>2</sub>O molecule shown here, it should hopefully reveal that there is not much to be gained beyond what is prescribed as “*tight*”.

Finally, we note that “*tier 1*” does not necessarily mean the same level of convergence accuracy for all elements. For light elements, *tier 2* may often be needed, and we set them by default in our *tight* settings. For significantly heavier elements (transition metals in particular), *tier 1* is already well converged for ground-state DFT calculations, which is therefore mostly the default in our *tight* settings.

## 4.2 Heavy elements ( $Z \gtrsim 30$ ): Modifications for scalar relativity

With H and O, the preceding, simple H<sub>2</sub>O testcase involves only elements so light that relativistic effects on their total energies can be safely ignored in production calculations.

The rule of thumb that relativistic effects do not matter much for quantities derived from total energies (binding energies, geometries) holds up to elements number  $Z=20$ -30 (Ca or Zn), depending on the required accuracy. For example, the DFT-LDA lattice parameter for fcc Cu is 3.54 Å in the non-relativistic case, but 3.52 Å if relativistic effects are included. In any case, relativistic effects should be accounted for in accurate calculations *beyond* these elements. As described in more detail in Ref. [12], this process is handled by FHI-aims at different levels of approximation, with some overhead resulting compared to the simple non-relativistic case.

In a nutshell, the physical impact of relativity for heavy elements *is* noticeable not just as a total-energy offset, but importantly in total energy differences, and in particular impacts also geometries. The underlying reason is that core and valence orbitals “see” the near-nuclear region (where relativistic effects are most important) differently, depending on their angular momentum  $l$ . Somewhat simplistically put, the increased relativistic mass of the electron near the nucleus results in a tendency for all orbitals to “shrink” compared to the non-relativistic case, but to a different degree for different orbitals. The shrinkage thus changes not just atom sizes, but also the nature of bonding itself. A detailed discussion of relativistic effects is beyond the scope of this manual (and can be found in many excellent reviews, e.g., Ref. [94]) – but bear in mind that relativistic effects should not simply be shrugged off!

As outlined in Ref. [12] and Section 3.8, FHI-aims provides two levels of approximation to scalar relativistic effects: the “atomic” and “scaled” zero-order regular approximation (ZORA). Both are invoked by the `relativistic` keyword. In our experience, energy differences and geometries from *both* approximations closely match relativistic benchmark calculations performed using the full-potential linearized augmented plane wave method. However, the absolute total energies from scaled ZORA are significantly closer to the fully relativistic limit than atomic ZORA. On the other hand, the implementation of total energy *gradients* is much simpler in the atomic ZORA case.

We therefore recommend the following strategy to perform standard relativistic calculations with FHI-aims:

- For calculations involving forces and geometry changes (structure optimization, molecular dynamics, vibrations), the recommended option is `“relativistic atomic_zora scalar”`.
- For final single-geometry energy differences (converged geometries, reaction barriers, ...), perform a second, independent calculation using `“relativistic zora scalar threshold”` (where threshold is a small number, e.g.,  $10^{-12}$ ).

With this strategy, all final results will be obtained at the “scaled ZORA” accuracy level,

close to the best alternative scalar-relativistic methods available. Before turning to a specific example, we emphasize two important points for the practical use of the “scaled ZORA” approximation:

1. **Only use the *final* total energy from a converged scaled-ZORA run**, given after the scaled ZORA post-processing step is complete. During the s.c.f. cycle, the Kohn-Sham wave function and all intermediate total energies are obtained at the simple ZORA level only, so *any intermediate total energy is not reliable*.
2. **For energy differences between different structures, use the same `radial_base` integration grids**. Relativistic effects near the nucleus need to be integrated accurately, but residual integration errors cancel on an atom-by-atom basis. In energy differences, the remaining errors for the same `radial_base` and `radial_multiplier` settings thus cancel out.

We illustrate the practical use of of atomic and scaled ZORA for the Au dimer in DFT-LDA, found in the `testrun/Au_dimer` directory.

In the subdirectory `relax_light`, a quick but sufficiently accurate prerelaxation is set up, at the `atomic_zora` level. The `control.in` file is set up to use `relativistic` `atomic_zora` scalar and `relax_geometry` bfgs `1.e-2` to converge forces down to  $10^{-2}$  eV/Å. Since this is intended to be a quick prerelaxation run (starting with an arbitrary separation of 3 Å of both atoms), the “light” species default settings for Au are used for the `species` subsettings. Compared to the much more accurate “tight” settings,

- the radial and angular integration grids are significantly reduced,
- the Hartree potential expansion is capped at `l_hartree=4`,
- the cutoff potential onset and width in `cut_pot` are reduced to a minimum that we still consider safe (3.5 Å / 1.5 Å, respectively),
- the basis set is the `spdf` section of `tier 1` only.

. Among these settings, the reduction of the basis set to `spdf` has the biggest impact on the resulting equilibrium geometry,  $d=2.465$  Å. Just adding the `g` function of `tier 1` would already reduce the error by 0.013 Å.

In the subdirectory `postrelax_tier1`, the final geometry from the previous step is used as the starting point for an accurate post-relaxation using the “tight” `species_defaults` settings. Note that, while all other settings are tightly converged at this level, the basis set convergence should normally still be tested explicitly. The `tier 1` level used here for Au is quite sufficient for an accurate result. The binding distance converges after two additional relaxation steps, at  $d=2.449$  Å.

In the subdirectory `final_scaled_zora`, the fixed post-relaxed equilibrium geometry is used to compute the Au<sub>2</sub> binding energy at the scaled ZORA level. The computation is a single-point run with the full `tier 1` basis set, using the rather safe grid and Hartree potential settings from the “tight” species default file. FHI-aims output files using the

same `control.in` file are given for both  $\text{Au}_2$  and a single Au atom, yielding a binding energy of 3.263 eV for the dimer. Note that this deviates from a possible `atomic_zora` result by less than 0.02 eV, matching our experience that energy *differences* usually agree very well between `atomic_zora`, scaled `zora`, and independently with results other scalar relativistic all-electron results obtained using other codes.

In conclusion, this section illustrates how a quick pre-relaxation followed by a safe calculation of the relevant energy differences can be combined. The key point in this endeavour is to ensure that the final relaxed geometry is accurate with as little computational overhead as possible.

## 4.3 *k*-point sampling in the Brillouin zone for semiconductors

In a semiconductor, the choice of the *k*-grid is crucial for accurately sampling the Brillouin zone, because the valence band maxima (VBM) and conduction band minima (CBM) are usually located at high symmetry points. Here, we describe three different choices for a *k*-grid offset:

- a  $\Gamma$ -centered grid,
- the grid suggested by Monkhorst and Pack [78] and
- an off- $\Gamma$  grid.

They are defined as

$$u_r = \begin{cases} \frac{r-q}{q} & \Gamma\text{-centered,} \\ \frac{2r-q-1}{2q} & \text{Monkhorst-Pack,} \\ \frac{r-1}{q} + \frac{1}{2q} & \text{off-}\Gamma \end{cases} \quad (4.1)$$

where  $r = 1 \dots q$  and  $q$  is the total number of points. In Fig. (4.2, 4.3 and 4.4) the three different shifts are illustrated. The Monkhorst-Pack grid<sup>1</sup> agrees with the  $\Gamma$ -centered grid for an uneven number of grid points and with the off- $\Gamma$  grid for an even number of points. The  $\Gamma$ -centered grid and sequentially the Monkhorst-Pack grid with an uneven number of grid points include the  $\Gamma$ -point.

To achieve the relevant settings in the `control.in` file, two keywords have to be changed. For the number of grid points the keyword `k_grid` needs to be specified:

`k_grid n1 n2 n3`,

where the numbers  $n_1$ ,  $n_2$ ,  $n_3$  are integers defining the number of splits along the reciprocal vectors.

For the off- $\Gamma$  and Monkhorst-Pack shift, a second keyword has to be given in the `control.in` file:

`k_offset f1 f2 f3`,

where `fi` are fractional coordinates between zero and one. The offset can be defined according to Eqn. (4.3).

$$\mathbf{k\_offset} = \begin{cases} \begin{pmatrix} 0.0 & 0.0 & 0.0 \end{pmatrix} & \Gamma\text{-centered,} \\ \begin{pmatrix} \frac{1}{2} - \frac{1}{2n_1} & \frac{1}{2} - \frac{1}{2n_2} & \frac{1}{2} - \frac{1}{2n_3} \end{pmatrix} & \text{Monkhorst-Pack,} \\ \begin{pmatrix} \frac{1}{2n_1} & \frac{1}{2n_2} & \frac{1}{2n_3} \end{pmatrix} & \text{off-}\Gamma \end{cases} \quad (4.2)$$

<sup>1</sup>One could argue that Monkhorst and Pack never intended their formula to be used for odd  $q$ . Still, for the sake of this section, “Monkhorst-Pack” refers to this definition.

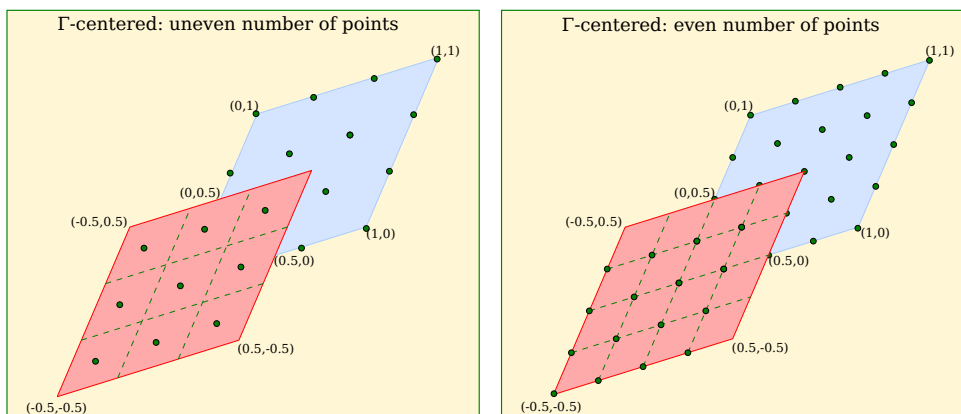


Figure 4.2: The  $\Gamma$ -centered grid in two dimensions for even and uneven numbers of grid points. In both cases the  $\Gamma$ -point is included in the grid.

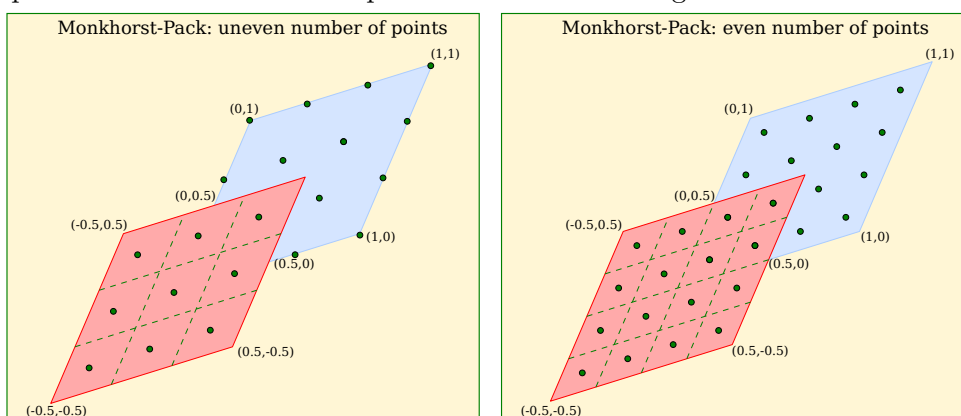


Figure 4.3: The Monkhorst Pack grid [78] in two dimensions for even and uneven numbers of grid points. In the left picture (uneven number of grid points) the  $\Gamma$  point is included in the grid.

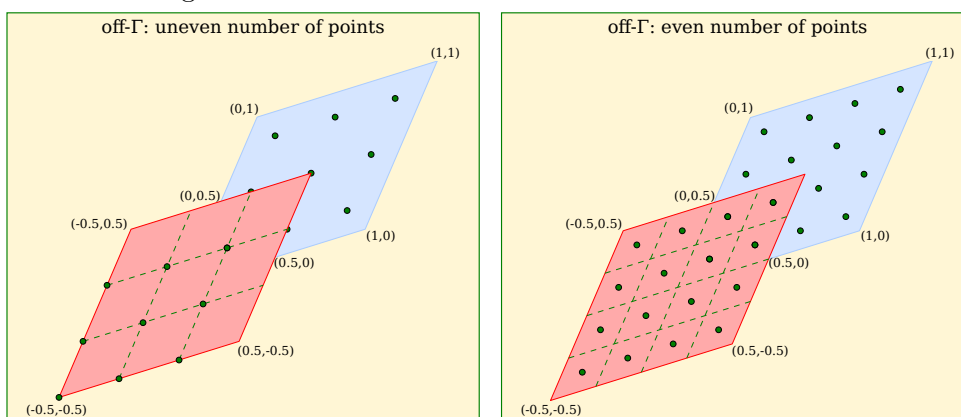


Figure 4.4: The off- $\Gamma$  grid in two dimensions for even and uneven numbers of grid points.

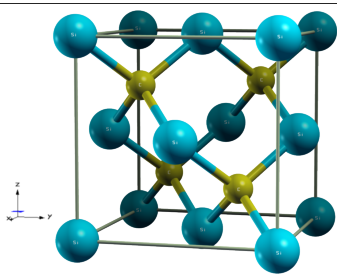


Figure 4.5: The crystal structure of silicon carbide in its cubic zinc blende structure (3C-SiC)

The oscillatory behaviour in Fig. (4.6) of the Monkhorst-Pack grid reflects the agreement with the off- $\Gamma$  grid in the case of an even number of grid points and the  $\Gamma$ -centered for uneven number of grid points respectively.

Here, we present a concrete example. Cubic silicon carbide (3C-SiC) is a semiconductor featuring an indirect band gap between the  $\Gamma$ -point and the  $X$ -point [see Fig. (4.7)]. In Fig. (4.5) the crystal structure of 3C-SiC is shown. The crystal is similar to the zinc-blende structure. The lattice constant was found to be  $4.36 \text{ \AA}$  [53]. The  $k$ -grid was tested in terms of grid density in the Brillouin zone and the three different shifts discussed above.

As a reference energy the total energy was taken with a  $k$ -grid of  $25 \times 25 \times 25$   $k$ -points. In Fig. (4.6) the energy differences  $|E(k=25) - E(k)|$  are plotted on a logarithmic scale versus the number of  $k$ -points.

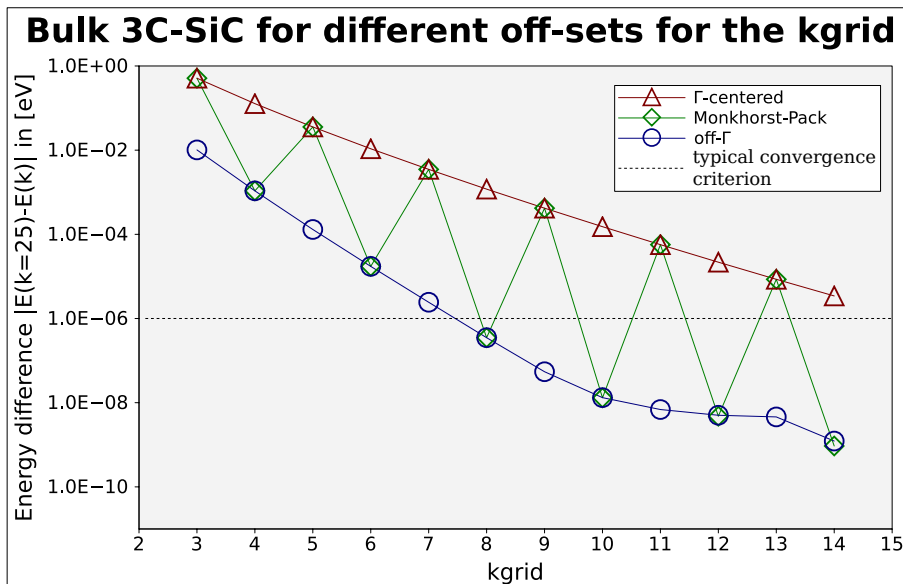


Figure 4.6: The  $k$ -grid was tested with respect to the grid density and the shift of the grid in reciprocal space. The shifts are defined in Eqn. (4.1).

A typical convergence criterion for the total energy is a value of  $10^{-6}$  eV. In Fig. (4.6) the dashed line marks the convergence criterion. In the case of an off- $\Gamma$  shift the calculation is well converged with a  $k$ -grid of  $8 \times 8 \times 8$ . For a  $\Gamma$ -centered grid the calculation did not reach convergence with a grid of  $14 \times 14 \times 14$ . As mentioned before, in 3C-SiC the VBM is in the  $\Gamma$ -point. Therefore the convergence of the total energy with respect to the density of the  $k$ -grid is slow for every grid including the  $\Gamma$ -point, because the contribution of the energy in the  $\Gamma$ -point is over-weighted. In this case the off- $\Gamma$  shift gives best convergence.

We note the above conclusion is mainly applicable for semiconductors. In a metal, the sampling of the Fermi-surface is crucial, this demands a high  $k$ -grid density rather than a special off-set.

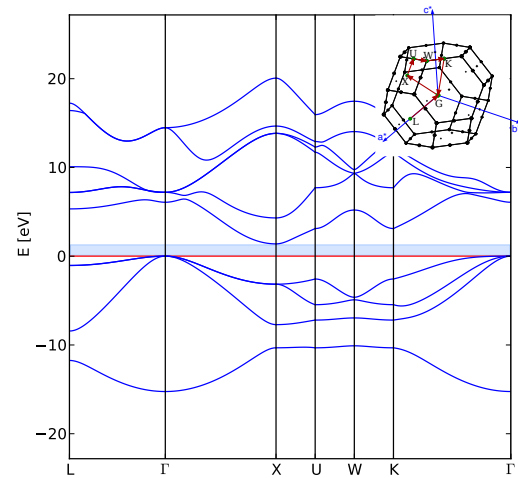


Figure 4.7: The band structure of 3C-SiC with tight, Tier 2 basis settings.

```

#
# plotting bands from FHI-aims!
#
# found reciprocal lattice vector   1.784996   1.784996  -1.784996
# found reciprocal lattice vector  -1.784996   1.784996   1.784996
# found reciprocal lattice vector   1.784996  -1.784996   1.784996
#
# Starting point for band 00,      W = ( 0.89250,  0.00000,  1.78500) \
will be at real distance =      0.00000
# Ending   point for band 00,      L = ( 0.89250,  0.89250,  0.89250) \
will be at real distance =      1.26218
#
# Starting point for band 01,      L = ( 0.89250,  0.89250,  0.89250) \
will be at real distance =      1.26218
# Ending   point for band 01, Lambda = ( 0.44625,  0.44625,  0.44625) \
will be at real distance =      2.03511
#
# Starting point for band 02, Lambda = ( 0.44625,  0.44625,  0.44625) \
will be at real distance =      2.03511
# Ending   point for band 02, Gamma = ( 0.00000,  0.00000,  0.00000) \
will be at real distance =      2.80803
#
[...]
```

Figure 4.8: Verbatim header of the band structure plotting procedure for FHI-aims. Note the position of the high-symmetry points along the real axis of the bands.

## 4.4 Plotting the band structure and density of states

*Note: The fcc Cu test case described in this chapter was replaced by fcc Al in the current FHI-aims distribution. For fcc Al, the example should work just as for fcc Cu, but the output figures will not be the same. Apologies for the omission—this will be corrected in a future version of the manual.*

An example for plotting band structures and the density of states is contained in the example for *fcc-Copper*, where the appropriate output lines simply have to be uncommented. This section contains a walk-through of the necessary steps and keywords to obtain this band structure.

To get an output of the band structure, use the keyword `output band`. The completed calculation of a given set of bands yields the files `band1XXX.out`, which are the starting point for the script `aims_band_plotting`, provided with this distribution. To get a complete output of the band structure, change into the directory of the calculation which must also include the original `control.in`, and run the plotting script. The *fcc-Cu* example yields the header shown in Fig. 4.8.

Apart from reorganizing the data from the output files into columns that only contain the *scf*-eigenvalues, the main purpose of the `aims_band_plotting` script is to provide the proper scale of the Brillouin zone and to provide the location of all the special points

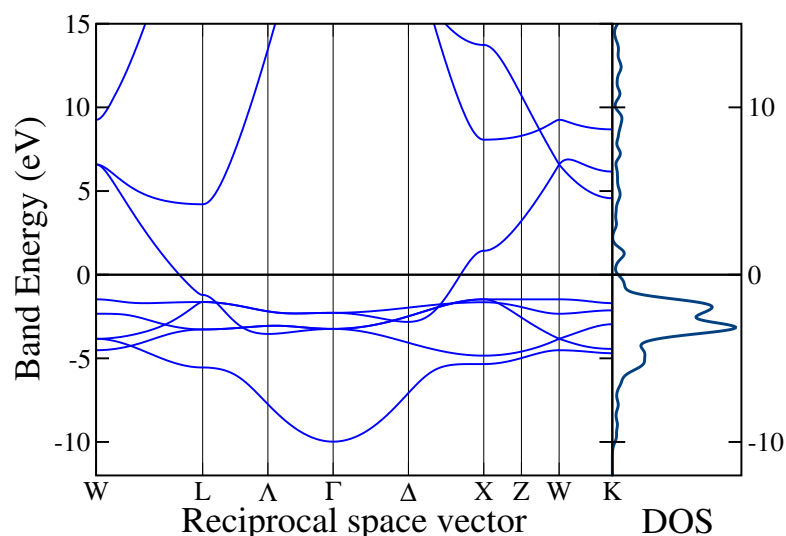


Figure 4.9: Band structure and total density of states for the Cu-*fcc* structure provided in the example runs.

mentioned in the `control.in` along the linear plot, as shown in Fig. 4.8. The format of the numerical output following the header is

```
x band1 band2 band3 band4 ... bandN,
```

where  $x$  is the position along the  $k$ -axis and  $\text{band}_i$  is the energy of the  $i^{\text{th}}$  band at the given  $k$ -point, in units of eV. Note that FHI-aims outputs *all* bands it has computed, in the case of an all-electron code this naturally includes the core levels as well. The number of unoccupied bands can be adjusted with the keyword `empty_states`. The properly scaled output for the band structure is shown in Fig. 4.9.

To obtain a density of states plot, there are currently four different options.

- the total density of states, using the keyword `output dos`. There is an additional option to calculate perturbative density of states, using the keyword `dos_kgrid_factors`. This affords a much denser mesh for an accurate description of the density of states (see section 3.22).
- an atom-projected density of states, using `output atom_proj_dos`
- an angular-momentum component projected density of states averaged over all atoms of a *single species*, using the keyword `output species_proj_dos`. This is useful because different atoms of the same element may be grouped together as a separate species in `control.in`, e.g., all atoms in the first layer of a slab.

The syntax is the same for all three output options and is described in detail in section 3.22, see e.g. `output dos`. As there are various conventions for plotting the density of states, each possible DOS plot produces two different output files. The first one, ending in “\_raw.dat”, contains the exact density of states, using exactly the same eigenvalue energies and energy zero as internally in FHI-aims (in cluster systems, the energy zero is simply the vacuum level; in periodic systems, the energy zero is set by the  $\mathbf{G}=0$  component of the long-range electrostatic potential). The second file contains the same

information, but the energy zero is shifted to the Fermi-level (metallic systems) or the valence band edge (insulators) of the system. Fig. 4.9 shows the density of states for the example of same *fcc* Cu in the *testrun* directory.

Note that there are two useful projections of the DOS, both of which are based on a Mulliken analysis. `output species_proj_dos` can be used to get a species-dependent angular momentum projection, while `output atom_proj_dos` resolves the states into the various atomic contribution. This is useful, for example to reconfirm that the valence band in the example calculation are indeed almost completely composed of *d*-bands (an exercise recommended to the reader), or to figure out which atoms a given level resides on.

## 4.5 Visualizing charge densities and orbitals

Charge densities, orbitals, etc. can be written onto a cartesian grid in the “cube” file format (see below, as well as Section 3.22, specifically the `output` keyword and the `cube` subkeyword described there). The visualization described in this section pertains to a simple H<sub>2</sub>O molecule, and can be repeated for example for the relaxed geometry obtained in the relaxation testrun for H<sub>2</sub>O in this distribution.

After running FHI-aims for the desired target geometry with the appropriate output flags (see Section 3.22), some files will be generated, all of which have the extension “\*.cube”. In the H<sub>2</sub>O example shown in this section, the calculation was run with *tier 2* basis set, which should produce a more accurate charge density, and the parameters entered in the control.in file were:

```
output cube total_density
  cube origin 0.0 0.0 0.0
  cube edge 29 0.15 0.0 0.0
  cube edge 29 0.0 0.15 0.0
  cube edge 29 0.0 0.0 0.15
output cube eigenstate 5
output cube eigenstate 6
```

The “Gaussian CUBE” files are written in a file format originally defined by the “Gaussian” program package, but now implemented as a *de facto* standard by many visualization programs. In this section, an example of how to visualize them using jmol program [55] is presented.

The CUBE files are composed of a header and the volumetric data. The header is divided in the following manner:

- 1<sup>st</sup> and 2<sup>nd</sup> lines: text ignored by visualization programs
- 3<sup>rd</sup> line: number of atoms in the file followed by the origin around which the volumetric data will be plotted.
- 4<sup>th</sup>, 5<sup>th</sup>, and 6<sup>th</sup> lines: number of points to be plotted, followed by the axis vector.
- 7<sup>th</sup> line on, until the end of the header: one line for each atom of the system, consisting of atomic number, followed by the atom *xyz* position.

An example of such a header with the beginning of the volumetric data is given below:

```
CUBE FILE written by FHI-AIMS
*****
  3  -2.100000  -2.100000  -2.100000
 29   0.150000   0.000000   0.000000
 29   0.000000   0.150000   0.000000
```

```
29  0.000000  0.000000  0.150000
 8  0.000000  0.000000  0.000000  0.000000
 1  0.000000  0.707000 -0.707000  0.000000
 1  0.000000 -0.707000 -0.707000  0.000000
0.12810E-04 0.18208E-04 0.25394E-04 0.34819E-04 0.46974E-04 0.62322E-04
0.81196E-04 0.10365E-03 0.12928E-03 0.15704E-03 0.18520E-03 0.21135E-03
0.23277E-03 0.24687E-03 0.25180E-03 0.24687E-03 0.23277E-03 0.21135E-03
.
.
.
```

In order to visualize these files in jmol, open the program and open the script console. Load a cube file, for example the total electronic density, by typing the following command:

```
> load "total_density.cube"
```

This will show your molecule. In order to visualize the surfaces, one must use the "isosurface" command. It has a myriad of options, all of which are explained in the jmol documentation [55], as of this writing located at <http://chemapps.stolaf.edu/jmol/docs/#isosurface>. As an example, in order to plot a volume, the command is:

```
> isosurface cutoff <cutoff> "total_density.cube"
```

The field <cutoff> specifies a radius for the surface, since all values equal too or less than this one will be plotted.

Although these files are written by default in Å, some programs (including jmol), read them in atomic units (bohr) by default. In the *utilities/* folder, one can find the *angstrom\_to\_bohr.pl* script that converts the CUBE files to atomic units.

Plotting planes is also very easy, and the command is, for example:

```
> isosurface plane <plane_position> <other_options> "total_density.cube"
```

The field <plane\_position> can either specify the axis that define your plane (e.g. xy) or three atoms, whose centers will specify the plane (syntax: (atomno=1) (atomno=2) (atomno=3)). The field <other\_options> may contain all sorts of color scheme and/or cutoff specifications.

One can plot as many isosurfaces simultaneously as one wishes. The command to delete them is simply:

```
> isosurface delete
```

Some example images and their respective commands are shown in Figure 4.10. The CUBE files shown were all converted to atomic units.

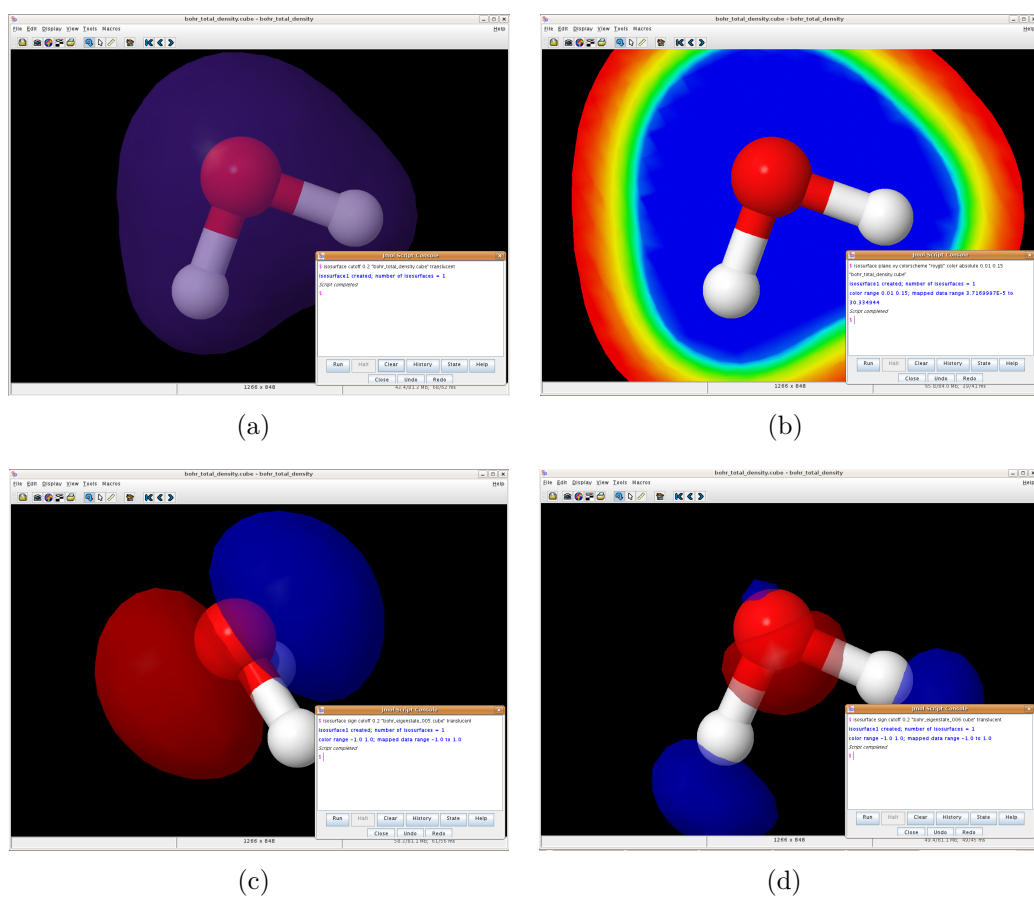


Figure 4.10: Jmol plots of charge densities and orbitals.

## 4.6 Calculation of vibrational and phonon frequencies

Vibrations (for non-periodic structures) and phonons (for periodic structures) can be computed in FHI-aims via script-based finite difference approaches. Each tool first does all necessary DFT calculations and then calls a routine to set up and diagonalize the Hessian matrices.

Technically, the non-periodic and periodic versions use separate sets of scripts. In addition, we now support two different sets of scripts for the phonon case:

- Infrastructure for vibrations in non-periodic systems (clusters or molecules). The relevant perl scripts and source code for this step for the entire finite-difference calculation are included with FHI-aims. To compile the tools, change into the source directory from which you also compiled the original FHI-aims. With the same Makefile settings as for the main program, type either `make vibrations` or `make vibrations.mpi`.
- Infrastructure phonons in periodic systems, contained *within* the FHI-aims distribution. The make targets for the phonons are `phonons` and `phonons.mpi`. This version is based on perl scripts to obtain the relevant input data from FHI-aims etc., and Fortran-based code for the numerical work. The advantage of this infrastructure is that it should work out of the box. Note, however, that the external `phonopy` package described below has a greater level of sophistication, including some features that are not available within the FHI-aims internal infrastructure.
- Infrastructure phonons in periodic systems, based on the GPL `phonopy` tool (not distributed with FHI-aims itself). `Phonopy` is a python-based utility to generate and process finite-difference generated phonon-related quantities from a variety of codes. The infrastructure for FHI-aims support in `phonopy` was mainly written by Jörg Meyer of FHI, and contributed to the GPL'd `phonopy` project. We note that `phonopy` and its dependencies must be installed separately (in addition to FHI-aims) on your system, and we do not officially provide support for it. That said, it is pretty certain that reasonable questions will be answered in the FHI-aims user forums, since a number of people use the `phonopy` infrastructure with FHI-aims for production work.

In the following, we focus on the tools shipped with FHI-aims. Some additional remarks on the `phonopy` infrastructure can be found in a subsection near the end of this section. Note that the `phonopy` version uses the same keywords as the FHI-aims internal phonon infrastructure for `control.in`, so reading the entire section is certainly not useless.

For the FHI-aims internal tools (vibrations and phonons), we recommend that MPI-routines be compiled into these binaries whenever an MPI version of the code is being used. The Make targets will create two files in the FHI-aims binary directory. One is the actual diagonalization subprogram, the other one the `perl-wrapper`. **You will need to edit the header of the `perl-wrapper` scripts** and give three key parameters: The absolute location of the FHI-aims binary directory (`$AIMS_BINDIR`), the aims executable

to be used (\$EXE), and the calling command for the executable (\$CALLING\_PREFIX and \$CALLING\_SUFFIX). The \$CALLING\_PREFIX is required for specification of parallel runs, for example

```
$CALLING_PREFIX = mpirun -np 2
```

for a two-core parallel run. In addition, some parallel environments (most notably IBM's poe require that the number of cores is specified after the executable is called, for those instances you can set \$CALLING\_PREFIX. Note that the postprocessing routines for both, the vibrations and the phonons are also parallel. Specifically, the calculation of the phonon density of states makes good use of that functionality.

Having compiled the vibrations and/or phonon scripts, running them is very simple and will be demonstrated by two examples. Both are provided with the code distribution.

## Vibrations of a water molecule

We use the same water molecule that was relaxed in the example section 4.1. This test case is contained in the directory `testrun/H2O-vibrations`.

We are here calculating a second derivative of the energy from the numerical change on the forces arising from small finite displacements  $\delta$ , in the following way:

$$\frac{\partial E}{\partial x_i \partial x_j} = \frac{\mathbf{F}_i(x_j + \delta) - \mathbf{F}_i(x_j - \delta)}{2\delta} \quad (4.3)$$

Here,  $E$  is the total energy as a function of all atomic coordinates  $x_i$  ( $i=1, \dots, 3M$  for  $M$  individual atoms), and  $\mathbf{F}_i$  are the components of the analytical first derivatives (forces) on each atom (again,  $i=1, \dots, 3M$ ) due to a displacement of coordinate another  $x_j$  by an amount plus or minus  $\delta$ .

Since we are interested in the Hessian in a local minimum of the potential energy surface (ideally,  $\mathbf{F}_i(\{x_j\})=0$  for all  $i$  at the local minimum geometry  $\{x_j\}$ ), this expression is the difference of two force values that are themselves already near zero. The smaller the displacement  $\delta$ , the smaller the absolute magnitude of the forces to be subtracted, giving greater weight to any residual numerical noise in the calculation. The larger the displacement  $\delta$ , the larger does the difference become, but at the same time, we will also move out of the region of the potential energy surface that is exactly harmonic, and introduce systematic errors into the Hessian. We are thus faced with a tradeoff between choosing a value  $\delta$  that is large enough to be free of any potential numerical noise, yet small enough to avoid large systematic errors due to real anharmonicities. The default value for  $\delta$  is 0.0025 Å, but this value should be checked explicitly in any practical calculation.

Since the vibrational frequencies depend strongly on the accuracy of the forces, and on any near-zero residual forces on the fully relaxed geometry itself, we apply the following changes to the earlier file `control.in`:

- We use the *tight* species defaults for elements H and O. Among other things, these defaults prescribe denser integration grids than *light* and thus lead to more

accurate forces. Obviously, the previous optimum geometry from the *light* settings should be postrelaxed to the exact minimum for the modified *tight* settings, and our script does this automatically (but check the output just in case).

- We increase the accuracy to which the forces are converged in the self-consistency cycle, `sc_accuracy_forces`, to 1E-5. *However*, be sure to check that the remaining settings, especially `sc_accuracy_eev` are accurate enough. You do *not* want your FHI-aims calculation to spend more than 1-2 s.c.f. cycles per geometry on actually converging the forces, because each s.c.f. iteration with forces can be up to a factor 10 more expensive than an s.c.f. iteration during which the forces are not yet checked.
- We change the `relax_geometry` bfgs convergence criterion to 1E-4. This is, again, a rather harsh requirement, and one should verify (especially for large molecules) that the relaxation algorithm is not spending large amounts of time on failed relaxation steps near the minimum, probing any residual numerical noise of integration grids etc. rather than following an actual, smooth potential energy surface.

These new settings contain all that is necessary to get close enough to the actual minimum. Again, the convergence settings quoted above should normally be fine, but in case of doubt, check. If, for some reason, those stringent criteria can not be reached exactly, it is still preferable to obtain a slightly less converged Hessian within an amount of CPU time that is actually finite.

In order to run the calculation of the vibrations, change to the directory containing the input files `control.in` and `geometry.in` and run the `aims.vibration.*.pl` script in the directory. That's it. The calculation should take no more than a few minutes to complete. It first relaxes the structure to its minimum configuration and then applies six finite displacements to each of the atoms: 18 single-point calculations in total for the water molecule. The result (using the default settings for  $\delta$ ) can be visited in subdirectory `test_default`.

The output stream contains all of the important data, which is additionally saved in a number of temporary and output files. The file `basic.vib.out` contains the output of the normal mode analysis, while the file `basic.xyz` contains the actual eigenmodes (vibrations), which can be read by a number of molecular viewers.

A short note on the actual physical output `basic.vib.out`: The frequencies are given in  $\text{cm}^{-1}$ , the corresponding zero point energies in eV, and their cumulative total. It is important to ensure that the first six eigenmodes are close to zero, which means that the structure in question actually corresponds to a minimum on the potential energy surface. FHI-aims does not do an *a priori* reduction of the translational and rotational eigenmodes, which allows an explicit check on the quality of the structure.

In the default version (no command line options specified), the vibration script automatically assumes a jobname `basic`. This name, as well as the finite difference displacement  $\delta$ , can be changed on the command line, leading to the following complete calling form of the script:

```
aims.vibrations.*.pl <jobname> delta
```

`<jobname>` is a name of choice that will be appended to all output files produced during the run. One can thus run the same job with different settings  $\delta$  multiple times in the same directory, starting from the same input files.

The second parameter, `delta`, is the finite displacement  $\delta$  used for calculating the Hessian matrix. Its default is 0.0025 Å, which has been doing a good job for all the cases we are aware of, but still, please verify that it works for your particular system. (See the brief discussion above).

For example, for our test case, subdirectory `test_delta_0.001` contains a second example run with the following command line:

```
aims.vibrations.*.pl test_0.001 0.001
```

For the default settings, our resulting frequencies look like this:

Mode number	Frequency [cm <sup>-1</sup> ]	Zero point energy [eV]	IR-intensity [D <sup>2</sup> /Ang <sup>2</sup> ]
1	-12.66044778	-0.00078485	1.89406419
2	-0.15559990	-0.00000965	0.00266735
3	-0.00081379	-0.00000005	0.00000000
4	0.04885082	0.00000303	0.00002821
5	6.58934967	0.00040849	0.00000000
6	7.01167236	0.00043467	5.41560478
7	1592.96295435	0.09875111	1.63341394
8	3708.86739272	0.22992045	0.04298854
9	3813.74446188	0.23642200	1.07616306

In contrast, the smaller value  $\delta=0.001$  Å produces the following results:

Mode number	Frequency [cm <sup>-1</sup> ]	Zero point energy [eV]	IR-intensity [D <sup>2</sup> /Ang <sup>2</sup> ]
1	-5.13719038	-0.00031847	1.89407621
2	-0.06616895	-0.00000410	0.00323346
3	-0.00073155	-0.00000005	0.00000000
4	0.01938144	0.00000120	0.00002697
5	2.47172391	0.00015323	0.00000000
6	2.70760068	0.00016785	5.41498492
7	1593.00852915	0.09875393	1.63340808
8	3708.82166272	0.22991761	0.04299176
9	3813.70936996	0.23641982	1.07615430

So, in this case, the rather tight relaxation and force convergence settings do produce a set of translations and rotations that are closer to zero for  $\delta=0.001$  Å than the default.

However, once again be warned that for larger molecules the same harsh convergence criteria may not be applicable, and a too small value  $\delta$  can in fact inflate any residual noise. So, a small value  $\delta=0.001$  Å or even less should not be blindly expected to improve numerical accuracy.

It should also be noted that the “physical” vibrational frequencies 7-9, above the six translations and rotations, remain completely unimpressed by the change of  $\delta$ , either way. This is generally true, and would even remain true for a yet larger value  $\delta=0.005$  Å, which would also be a reasonable default choice.

## Phonon calculations in general

*Notes regarding the FHI-aims internal phonon infrastructure: (i) The present phonon infrastructure works, but does not (yet) provide support for long-range electrostatic fields ionic compounds, which leads to splittings of the LO and TO branches at the  $\Gamma$  point. If your work deals with ionic compounds, this effect may be large. One alternative is to use the experimental support provided in the phonopy infrastructure. (ii) We also note that the FHI-aims internal version is not safeguarded against small, almost-zero force components and corresponding small numerical noise in the actually computed phonons. In fact, such noise may be an indicator that a tighter convergence of integration grids is required for the problem posed.*

To obtain proper phonon spectra some extra information is necessary. To calculate the phonons, we use the so-called *direct* or *supercell* method, based on the approximations that the interactions necessary to calculate the phonons are finite [see e.g. Ref. [85] and references therein for details of the method]. All phonon related functionality is driven with a single keyword `phonon`, specified in the `control.in`. Its various instances and options are described in this section. To actually run the calculations, the script `aims.phonons.*` has to be called in the directory containing your `control.in` and your `geometry.in` file. Eventually, it will produce three output files:

`phonon_band_structure.dat`, which contains the phonon band structure

`phonon_DOS.dat`, which contains the phonon density of states

`phonon_free_energy.dat`, which contains all thermodynamic quantities

The phonon calculation proceeds as follows. A single (user-specified) unit cell is extended a number of times in the direction of all three basis vectors. A finite displacement technique as in the vibrations is then used to gather the necessary force response, which then makes up a basic force constant matrix. All finite displacement structures are investigated for their symmetry (within the octahedral group), such as to minimize the computations required and also to obtain properly symmetrized forces where applicable. Finally, the force constant matrix is diagonalized for a number of selected reciprocal lattice points (i.e. along the requested bands) and the eigenvalues give the phonon spectra. Two things are absolutely vital:

- **converge your supercell size!** The minimum sensible cell is  $2 \times 2 \times 2$ , and it might be tempting to restrict a calculation to this size, but it will be almost certainly wrong for small unit cell sizes, as shown in the example below.
- **the geometry.in MUST contain only the primitive 1x1x1 cell of your lattice!** The phonon calculation copies its own supercell. If, by accident, your `geometry.in` already contains a  $3 \times 3 \times 3$  supercell and you request phonons for a  $3 \times 3 \times 3$  supercell on top of that, you will end up with a  $(3 \times 3 \times 3)^2 = 729$  atom calculation, which will almost certainly be too much for your computer...

The phonon script is invoked from a directory containing a `control.in` and a `geometry.in` file. As there are a number of working files involved in the calculation, a directory

phonon\_workdir is made in which all DFT calculations are done and where the post-processing happens. The advantage of such an approach is that the script can be stopped at any time without losing any information at all. In fact, once the DFT results are present, the entire script can be rerun multiple times to optimize the phonon output without having to redo the expensive DFT analysis over again. All the physical output is given either in standard out or as files in the directory from which the phonon script was invoked.

Without much further ado, here is the description of the commands required to calculate lattice vibrations:

---

**Tag: phonon** (control.in)

Usage: `phonon` [subkeywords and their options]

Purpose: Main driver keyword for the phonon calculation script. has the subkeywords `supercell`, `band`, `dos`, `displacement`, as detailed below.

---

**phonon sub-tag: band** (control.in)

Usage: `phonon band` kstart1 kstart2 kstart3 kend1 kend2 kend3 npoints startname endname

Purpose: To define a phonon band to be plotted explicitly. The remaining syntax is kept the same as for the `output` option `band`: the first six parameters describe the starting and ending point in RELATIVE  $k$ -coordinates, npoints is the number of points in the band, followed by the names of the special points for starting and ending the band.

As for the band structure output (see also section 4.4), there is no real need for naming the special points as far as FHI-aims or the phonon calculation is concerned, but we provide a translator script for the output, which then plots a 'nice' phonon band structure on a single axis and uses the names of these points.

---

**phonon sub-tag: displacement** (control.in)

Usage: `phonon displacement` delta

Purpose: To define the absolute displacement of each atom for the finite difference calculation.

Default: 0.001 Å

---

**phonon sub-tag: dos** (control.in)

Usage: `phonon dos fstart fend fpoints broad qdensity`

Purpose: governs the output of the phonon density of states.

This keyword produces a DOS output to the file `phonon_dos.dat`, which describes the DOS along a frequency axis in THz. `fstart`, `fend`, and `broad` give the starting and ending frequency as well as the broadening in units of THz, while `fpoints` is the number of points. Unlike the output option `dos`, here we also have to specify a  $k$ -point density `qdensity`, which does the BZ integration on a `qdensity`×`qdensity`×`qdensity` grid. Normally, the dynamic matrix only contains a very small number of elements (3 in the example described below), which means that one can choose `qdensity` quite high in order to get a well-converged density of states. Notice that this value might be changed by the program if the DOS and  $c_v$  calculations are requested at the same time. In that case, the higher of the two values for `qdensity` specified for the two calculations is chosen in order to minimize the computational effort. See also the `free_energy` description.

---

**phonon sub-tag: `free_energy` (control.in)**

Usage: `phonon free_energy Tstart Tend Tpoints qdensity`

Purpose: governs the calculation of the phonon free energy, the internal energy, and the specific heat for a given unit cell.

The phonon free energy and some derived quantities can be requested with this keyword. They are calculated between the temperatures `Tstart` and `Tend` with a total of `Tpoints` steps. A  $k$ -point density `qdensity` is also required, as the calculation involves an integration over the Brillouin zone. Notice that this value might be changed by the program if the DOS and free energy calculations are requested at the same time. In that case, the higher of the two values for `qdensity` specified for the two calculations is chosen in order to minimize the computational effort. See also the `dos` description. The output for this keyword is written to a file `phonon_free_energy.dat` in the main directory of the phonon calculation. The header for this file contains the constant value for the zero point energy of all phonons, given by

$$ZPE = \int d\omega g(\omega) \frac{\hbar\omega}{2} \quad (4.4)$$

Notice that the `phonon dos` is required to evaluate this equation, which means that the same infrastructure (hence the same `qdensity`) is used for both calculations.

The phonon free energy in the harmonic approximation (and without the total energy contribution of the unperturbed lattice) is given by

$$F(T, V, N) = \int d\omega g(\omega) \left( \frac{\hbar\omega}{2} + k_B T \ln[1 - \exp(-\hbar\omega/k_B T)] \right) \quad (4.5)$$

The units for the free energy, (and all energies of this output for that matter) are in eV/(unit cell).

The phonon internal energy is computed much in the same way as the other two energies,

it is given by

$$U = \int d\omega g(\omega) \left[ \frac{\hbar\omega}{2} + \frac{\hbar\omega}{\exp(\hbar\omega/k_B T) - 1} \right] \quad (4.6)$$

In order to calculate  $c_v$ , the following well-known expression is used.

$$c_v = \int d\omega g(\omega) \frac{(\hbar\omega)^2}{k_B T^2} \frac{\exp(\hbar\omega/k_B T)}{(\exp(\hbar\omega/k_B T) - 1)^2} \quad (4.7)$$

The output units for  $c_v$  are in  $k_B$  per unit cell.

One final output in the free energy keyword is the phonon entropy  $S$ , which is given in the form  $-TS$  by subtracting the total energy from the free energy.

**phonon sub-tag: frequency\_unit** (control.in)

Usage: `phonon frequency_unit` unit

Purpose: Allows specifying the frequency output for the phonons by setting unit to either  $\text{cm}^{-1}$  or to THz. Default is  $\text{cm}^{-1}$

**phonon sub-tag: supercell** (control.in)

Usage: `phonon supercell` n1 n2 n3

Purpose: To define a  $n1 \times n2 \times n3$  supercell in which to approximate the calculation of the dynamic matrix, based on the lattice vectors provided in `geometry.in`.

**phonon sub-tag: symmetry\_thresh** (control.in)

Usage: `phonon symmetry_thresh` thresh

Purpose: To define the maximally allowed coordinate difference (in Å) for the phonon symmetry checker to decide that two atoms are at the same location.  
Default:  $10^{-6}$

This keyword should not have to be changed from its default, but it might become useful when calculating phonons of larger supercells which have been internally relaxed: In that case, the symmetry constraints might have to be relaxed a little bit for atoms to be at the same position. However, be warned that this procedure has not been tested for its accuracy. Please do so before believing any of the results.

## phonopy and FHI-aims

As mentioned near the beginning of this section, it is also possible to use the interface *phonopy-FHI-aims* to the *phonopy code* (<http://phonopy.sourceforge.net/>) to perform

such phonon calculations. Some additional information can be found in the README text file provided at <http://www.fhi-berlin.mpg.de/aims/utilities.html> .

To use the *phonopy-FHI-aims* infrastructure,

python ( $\geq 2.5$ ) <http://www.python.org>

numpy <http://numpy.scipy.org>

and a C compiler, e.g., gcc

are required. The necessary symmetry analysis routines, which are taken from *spglib* (<http://spglib.sourceforge.net/>), are already included in the *phonopy* package. To obtain plots via the integrated plotting functionality,

matplotlib <http://matplotlib.sourceforge.net/>

has to be installed. The latter dependency is however not compulsory for running a calculation. Since the script completely decouples FHI-aims calculations and phonon pre- and post-processing operations (as detailed below) the intended usage is to run the script on a linux desktop, where all the mentioned packages can easily be installed via the package management system of the corresponding distribution, i.e. under Ubuntu linux:

```
apt-get install python python-numpy python-matplotlib
```

The *phonopy-FHI-aims* interface supports all input keywords and output styles discussed above. When called from the command line in a directory containing a correct `geometry.in` and `control.in` file, it generates  $n = 1, \dots, N$  subdirectories *phonopy-FHI-aims-displacement-n*. Each of these directories contains correct input files for one specific displacement. The calculations for the generated inputs need to be done manually (and concurrently if desired) on any available computing machinery. The output has to be stored in the respective subdirectory in a file named *basename.out*, e.g., *phonopy-FHI-aims-displacement-01/phonopy-FHI-aims-displacement-01.out*. Once the outputs are available (at the right places), rerunning the script in the same directory as before will produce the output (band structure, DOS, thermodynamic properties) which has been requested in `control.in`. The calculated data is written to ASCII files which include comments describing their contents. Some plots are also generated in `.pdf` files if matplotlib is available.

Additionally, the interface *phonopy-FHI-aims* supports some advanced features that are still experimental at this point:

- 'Matrix valued' supercells can be constructed.
- Corrections for the long-range electrostatic fields in ionic crystals ( $\Gamma$  point splitting of optical modes) can be accounted for. Please contact us if you need more information, as the required Born effective charges are not yet routinely computable within FHI-aims)

- The force constants for the whole supercell can be written out (a prerequisite to use the thermodynamic integration routines for anharmonic free energy calculations).

Again, some additional information can be found in the README text file provided at <http://www.fhi-berlin.mpg.de/aims/utilities.html> .

## 4.7 Transition state search: Nudged Elastic Band method

### 4.7.1 Theory and methods

### 4.7.2 Usage

The transition state search in AIMS at present is implemented in form of a PERL-script `NEB.pl` that calls a FORTRAN-90 code `NEB.f90`, both are located in the `src`-subdirectory `NEB`. The Fortran code still has to be compiled with LAPACK and BLAS routines linked in.

The job description of a TS-search is contained in a separate control file with a number of keywords defined below. That control file should be called `control_NEB.in`. You may change the name to anything else you like, just specify the name of the new control file as an argument when calling `NEB.pl`. Note that (at present) a lot of keywords are simply assumed to be there, and not enough testing is done to capture any missing options. This might lead to unexpected runtime behaviour. Please check to make sure everything you (think you) need is contained in the file `control_NEB.in`.

The interface to this program is via multi-frame xyz geometry files, which are described under the keyword `input_template`. You also need a control file template for the AIMS calls (see keyword `aims_control_file_template`), which must contain a line of the form

```
restart <aims_restart_string>
```

where the restart file for the different images can be specified and passed on. Towards the end of a run, this functionality greatly accelerates the DFT calculations. Also make sure that the AIMS control file template contains the two lines

```
compute_forces .true.
```

```
final_forces_cleaned .true.
```

The keywords used in the NEB-control file are:

---

**Tag:** `aims_control_file_template` (`control_NEB.in`)

- Usage: `aims_control_file_template` filename
- Location of the AIMS template file to be used for the TS search.

---

**Tag:** `aims_restart_string` (`control_NEB.in`)

- Usage: `aims_restart_string` template\_string
- Marker in the control file to be replaced by the actual restart file of a given iteration.

---

**Tag:** `dft_exe` (`control_NEB.in`)

- Usage: `dft_exe` filename
- filename is the AIMS executable including the full path.

**Tag:** `force_convergence_criterion` (`control_NEB.in`)

- Usage: `force_convergence_criterion` value
- value is the convergence force component on any image coordinate after it has been corrected by the transition state search.

**Tag:** `input_template` (`control_NEB.in`)

- Usage: `input_template` filename
- The location of the initial guess for the TS-search. The file name contains the number of the iteration, followed by `.xyz`, while `input_template` should be without those two points.
- Example: An initial guess called `space_dog_0.xyz` would be described with  
`input_template space_dog_`

- This file has to be a multi-frame xyz file with the following format:

```
<n_atoms>
start
<species1> <xstart> <ystart> <zstart>
...
<n_atoms>
image 1
<species1> <x1> <y1> <z1>
...
<n_atoms>
image <n_images>
<species1> <x1> <y1> <z1>
...
<n_atoms>
END
<species1> <x1> <y1> <z1>
...
```

**Tag:** `max_atomic_move` (`control_NEB.in`)

- Usage: `max_atomic_move` value
- The maximally allowed displacement during a single NEB step.
- This value should be relatively small (default is  $0.1\text{\AA}$ ) as the algorithm might

literally tear apart some of the images if the force constants are not set ideally and the transition path is not extremely close to the actual path.

- Same as keyword `max_atomic_move` in AIMS

**Tag:** `line_step_reduce` (`control_NEB.in`)

- Usage: `line_step_reduce` value
- for BFGS: factor by which a line step will be reduced if the object function increased more than `object_function_tolerance` between two successive iterations.
- at present only useful for `method` PEB, as there is no implementation of an object function for the other methods.
- This is the same as the original AIMS keyword `line_step_reduce`

---

**Tag:** `method` (`control_NEB.in`)

- Usage: `method` flag
- `flag` can be either PEB or NEB for the pure elastic band and the nudged elastic band methods respectively.

**Tag:** `min_line_step` (`control_NEB.in`)

- Usage: `min_line_step` value
- `value` is the minimal line step for the BFGS solver below which the code simply executes a step in order to get out of any numerical noise.
- same as the keyword `min_line_step` in the original AIMS code.

---

**Tag:** `n_atoms` (`control_NEB.in`)

- Usage: `n_atoms` value
- `value` is the number of atoms in each image.

---

**Tag:** `n_images` (`control_NEB.in`)

- Usage: `n_images` value
- `value` is the total number of images, not counting the start and end points.

---

**Tag:** `n_iteration_start` (`control_NEB.in`)

- Usage: `n_iteration_start` value
  - Default `n_iteration_start` = 0
  - If there is data available from a previous run of the same system, it might start at iteration value rather than from the beginning.
  - Careful that you use the proper `save_data` file if using this feature!
- 

**Tag:** `n_max_iteration` (`control_NEB.in`)

- Usage: `n_max_iteration` value
  - value is the maximal number of iteration of the search.
- 

**Tag:** `object_function_tolerance` (`control_NEB.in`)

- Usage: `object_function_tolerance` value
  - An iteration is rejected when the object function increases more than value between two successive iteration.
  - This only works for the pure elastic band `method` as there is no object function in the current NEB implementation.
  - Same function as keyword `energy_tolerance` in the main AIMS code.
- 

**Tag:** `save_data` (`control_NEB.in`)

- Usage: `save_data` filename
  - Gives a place to store BFGS and other information between successive iteration.
  - Be careful to delete this file when starting a new NEB run.
- 

**Tag:** `spring_constant` (`control_NEB.in`)

- Usage: `spring_constant` value
  - value is the single fixed spring constant throughout a single NEB run.
  - Using this is either default or it should be done in conjunction with `use_variable_spring_constants` `.false`.
- 

**Tag:** `spring_constant_max` (`control_NEB.in`)

- 
- Usage: `spring_constant_max` value
  - Maximal value in a range of possible spring constants
  - can only be used together with the option `use_variable_spring_constants` `.false.`
- 

**Tag:** `spring_constant_min` (`control_NEB.in`)

- Usage: `spring_constant_min` value
  - Minimal value in a range of possible spring constants
  - can only be used together with the option `use_variable_spring_constants` `.false.`
- 

**Tag:** `start_BFGS` (`control_NEB.in`)

- Usage: `start_BFGS` value
  - Force convergence criterion below which the algorithm switches from a steepest-descent type approach to BFGS algorithm.
- 

**Tag:** `switch_to_CI-NEB` (`control_NEB.in`)

- Usage: `switch_to_CI-NEB` value
  - will switch to using the climbing image nudged elastic band method after reaching a NEB force convergence of value
- 

**Tag:** `use_variable_spring_constants` (`control_NEB.in`)

- Usage: `use_variable_spring_constants` `.true./false.`
- If `.true.` a range of spring constants that depend linearly on the energy [46] is used.
- `.true.` Requires setting of the keywords `spring_constant_min` and `spring_constant_max`

## 4.8 Plugin for free-energy calculations with molecular dynamics: PLUMED

Molecular dynamics based free-energy calculations can be performed with the aid of the external plugin PLUMED.

Methods included are metadynamics [65], well-tempered metadynamics [8], umbrella sampling [107, 64, 99], Jarzynski-equation based steered molecular dynamics [54, 21]. A large and nearly exhaustive set of collective variable (CV) is accessible through a simple input script.

PLUMED is a free package that, after registration, can be downloaded from <http://merlino.mi.infn.it/~plumed/PLUMED/Home.html>. Currently, a copy of the PLUMED library is kept in the *external* directory of the FHI-aims source code, and must be compiled separately using the makefile `Makefile.meta` (see section 1.4). In the future a patch for modifying FHI-aims in order to compile it with PLUMED will be available on the PLUMED webpage.

### 4.8.1 Usage

The actual use of the plugin is switched on by this single line in `control.in`:

```
plumed .true.
```

With `plumed .false.` (default) or nothing, the code would behave exactly as compiled without this plugin. It is implied that some MD scheme must be used in `control.in`, in order to see PLUMED acting. What PLUMED does, in facts, is to modify the molecular dynamics forces according to the selected scheme.

All the specific controls of the free energy calculation are contained in the file `plumed.dat` (which must be in the working directory, together with `control.in` and `geometry.in`, if `plumed .true.` is set). For all the details on `plumed.dat`, we defer to PLUMED manual which can be found on the project website.

Here we report a minimal example for metadynamics:

```
PRINT W_STRIDE 10
DISTANCE LIST 1 <g1> SIGMA 0.35
g1->
2 3 4
g1<-
HILLS HEIGHT 0.003 W_STRIDE 10
ENDMETA
```

This script would make PLUMED deposit Gaussians (HILLS) of HEIGHT 0.003 hartree, every `W_STRIDE` timesteps. The (only) CV that will be biased by metadynamics is a distance between atom '1' and the center of mass of atoms '2', '3', and '4'. The number

labelling the atoms follows their order of appearance in `geometry.in`. The results will be printed (see below) every `PRINT W_STRIDE` time steps. The width of the Gaussian for the distance CV is specified by `SIGMA`

A note on the units: the units in `plumed.dat` and in the output(s) are the internal ones in FHI-aims, i.e. energies in hartree, distances in bohr, forces in hartree/bohr.

When using PLUMED, some extra output files are created. In `log.dat` the specifics of the run are given. `COLVAR` contains the trajectory of the selected CVs. Notably, if no biasing method is selected, but one or more CVs are defined in `plumed.dat`, PLUMED prints nonetheless the trajectory of those CVs in `COLVAR` (one can also explicitly switch off the biasing of *some* CVs via the `NOHILLS` directive).

In case metadynamics is used, then also `HILLS` is generated, which contains the informations for reconstructing the free energy profile. This is done with the postprocessing tool, “`sum_hills`”, which is given with the distribution.

For umbrella sampling a powerful tool for reconstructing the free-energy from `COLVAR`, can be downloaded from: <http://membrane.urmc.rochester.edu/Software/WHAM/WHAM.html>.

## 4.9 Script based parallel tempering (a.k.a. replica exchange)

A script based parallel tempering implementation is available. Part of the script is dependent on the particular batch-queueing system in use; with the distribution, we provide a solution that has been tested on linux machines with SGE batch-queueing system. Whereas the overall structure of the batch script would not change by changing the batch-queueing, few crucial lines might need intervention.

### 4.9.1 Usage

In order to run the parallel tempering the batch script “submit.rex” must be submitted to the queueing system. The batch script:

1. creates a subdirectory “rex\_??” for each replica,
2. copies the files needed for the FHI-aims run and runs them
3. manages the swaps between replicas.
4. prints outputs

The files that have to be present in the working directory are:

```
control.in.basic
control.in.rex
geometry.in.basic
optional: list_of_geometries
rex.AIMS.pl
submit.rex
```

The last two files are provided with the distribution and are contained in the subdirectory `utilities/REX`.

- `control.in.rex`, it must contain the following lines:
  - `n_rex` number of replicas
  - `temps` list of target T separated by a space; the number of T’s must agree with the above line
  - `freq` time interval between rex swaps, in ps, as in `control.in`
  - `MAX_steps` maximum number of replica exchange steps (i.e., the whole simulation will contain `MAX_steps*freq` ps per replica)
- `control.in.basic`, as in FHI-aims. Note, though, that the script will delete any keywords about geometry relaxation and MD, with the exception of `MD_time_step`, and appends at the end of each `control.in` in each subdirectory the `MD_settings` for the replica exchange. In detail, the following are the lines which are managed

by the script:

```
MD_run $t NVT_parrinello $temp[$i+1] 0.1
MD_MB_init $temp[$i+1]
MD_restart .true.
MD_clean_rotations .true.
output_level MD_light
```

where `$t` is a multiple of the “freq” keywords in `control.in.rex`, updated at each MD substep between swaps, and `$temp[$i+1]` is the target temperature for the particular replica and parallel tempering step. These lines are hard coded in the perl script `rex.AIMS.pl`.

- `geometry.in.basic`, written in the `geometry.in` format. It will be copied into each subdirectory, so that each replica would start from the same geometry.
- optional: `list_of_geometries` If present, it must contain a list of geometry files (each in the `geometry.in` format), one line each, that must be present in the working directory. The script will copy the file in the first line into the first subdirectory (i.e. related to the first temperature in `control.in.rex`), and so on. In case `list_of_geometries` contains less lines than the defined number of replicas, the “exceeding” replicas will start with the geometry contained in `geometry.in.basic`.
- `rex.AIMS.pl`, managing perl script. Nothing to be done here, in principle. If invoked as
 

```
perl rex.AIMS.pl stat <log_file>
```

 in a directory that contains a `log_file` created by `rex.AIMS.pl` itself (see next section), it provides useful statistics (even on the fly).
- `submit.rex` is the batch script. Some attention from the user is required here, too.
  - select the total number of slots with the keyword “# \$ -pe impi”, according to the number of replicas. The total number of slots is given by the desired number of replica times the desired number of slots per replica.
  - give the variable `type` the value ‘init’ or ‘restart’, according to the kind of run. Note that by running a ‘restart’, the script will complete the possibly interrupted parallel tempering steps (also only in some of the subdirectories) and then will continue with the replica exchange algorithm.
  - set the proper name and path for the `aims` binary

Below, the relevant area for the settings is reported:

```
##### to be taken care of by the user #####
binary='<binary path and name>'
# put type='init', if initializing, 'restart' if restarting
type='init'
# type='restart'
#####
```

## 4.9.2 Output

- in each of the subdirectories `rex_??` there are the files:
  - `temp.out` full FHI-aims output for the parallel tempering tempering step
  - `control.in` and `control.in`, the usual FHI-aims input files. They will change at each parallel tempering step, managed by the script.
  - `energy.trajectory`. Cumulative (i.e. appended after each attempted swap) energy trajectory for the replica.
  - `out.xyz`. Cumulative geometry trajectory, in xyz format.
- in the working directory: `log_rex`. It contains useful information on the swapping process. Below there is a commented example for a four replicas run.

```
> Mon Apr 5 03:51:05 CEST 2010
The time at the attempted swap
> Tt 100.0 200.0 150.0 250.0
The list of the running target temperatures, first place for rex_00, and so on
> map 1 3 2 4
Map of the temperatures in the "Tt" line, into the original list given in control.in.rex
> TE -6963471.3877 -6963471.2516 -6963471.4951 -6963471.3286
Total Energy ("Total energy (e1.+nuc.)") in each replica (first item in rex_00
and so on)
> swapping 3 1 @T 150.0 100.0 accepted
> swapping 4 2 @T 250.0 200.0 rejected
Detail of attempted swaps, with outcome
> temp 150.0 200.0 100.0 250.0
List of running target temperatures, after swaps.
> vfact 1.04880884817015 1 0.9534625892455937218 1
Rescaling coefficients for the velocities in each replica, for the next step
> ##### End of rex step #####
```

WARNING: when wall-clock ends in the middle of a prallel tempering step, it will always be printed the message:

```
WARNING: rex_??/temp.out Not converged?
Please check this problem before continuing.
If the reason that any of temp.out's does not reach not the end of the parallel
tempering step is the end of the wall-clock time, then the run can be safely
restarted by putting 'type=restart' in submit.rex
```

- in the working directory: `out.????`, where `????` is a temperature, in 4 digits. Constructed by appending the `temp.out` temporary outputs at the same temperature, each `out.????` contains the full FHI-aims output at the given temperature.

## 4.10 Formation energies of charged defects

The Gibbs free energy of formation of a defect is given by

$$\Delta G_f^D = E_{\text{tot}}^D - E_{\text{tot}}^{\text{perf}} - \sum_i n_i \mu_i^{\text{ref}} + q \varepsilon_F^{\text{ref}} - \sum_i n_i \Delta \mu_i + q \Delta \varepsilon_F, \quad (4.8)$$

where  $E_{\text{tot}}^D$  and  $E_{\text{tot}}^{\text{perf}}$  are the total energies of the defected and the perfect system,  $n_i$  is the number of atoms of type  $i$  added ( $> 0$ ) or removed ( $< 0$ ),  $\Delta \mu_i$  are the corresponding atomic chemical potentials referenced to  $\mu_i^{\text{ref}}$ ,  $\Delta \varepsilon_F$  is the Fermi level referenced to  $\varepsilon_F^{\text{ref}}$  and  $q$  is the charge of the system.

A common choice as a reference for the electron chemical potential  $\varepsilon_F^{\text{ref}}$  is the valence band maximum (VBM), so that the Fermi level can be assumed in the range between the VBM and the conduction band minimum (CBM), but in principle the choice of references for the chemical potentials is arbitrary.

FHI-aims uses the Ewald summation technique to calculate the electrostatic Hartree potential for a periodic system. For a charged periodic system (specified by the keyword `charge` in `control.in`) a neutralizing homogeneous background charge density is introduced to remove the divergent  $\mathbf{G} = 0$  component of the long-range part of the electrostatic potential.

This scheme is not suitable for periodic surface models, because the background charge density would be spread over the whole unit cell including the vacuum region. Instead charged surface defects can be treated within a virtual crystal approach (VCA), which corresponds to distributed doping of the material. The following scheme can be used for an insulating system with a localized defect level in the bandgap. By modifying the charge of the atomic nuclei (using the keyword `nucleus` in `control.in`), while keeping the system neutral, additional delocalized states can be introduced at the top of the valence band or at the bottom of the conduction band. The occupation of the defect levels can thus be tuned by the amount of charge distributed on the cations or anions in the system. To ensure that the defect has the desired charge  $q$ , the sum of the modified nuclear charges  $Z'_i$  should differ from the sum of the original nuclear charges  $Z_i$  by the value of  $q$ :

$$\sum_i^{N_{\text{atoms}}} Z'_i = \sum_i^{N_{\text{atoms}}} Z_i - q.$$

Note that for calculating the formation energy of a charged defect within the VCA the reference system should be the doped undefected system, not the perfect undoped system. Since doping pins the Fermi level  $\Delta \varepsilon_F$  vanishes for this method.

For example, a way to model a positively charged oxygen vacancy at a metal oxide  $\text{Me}_x\text{O}_y$  surface is to distribute the charge uniformly on the metal atoms  $\text{Me}$  by changing their nuclear charge from  $Z(\text{Me})$  to

$$Z(\text{Me}^{\text{VCA}}) = Z(\text{Me}) - \frac{q}{N(\text{Me})},$$

where  $N(\text{Me})$  is the number of metal atoms in the system. This introduces vacant states at the VBM which in the defected system will be occupied by electrons from the defect level.

When using the neutralizing background method for bulk systems the additional term may introduce an arbitrary shift, so that it is necessary to find a common energy reference for the charged and the neutral periodic system to which the respective potentials can be aligned. For example alignment of the core levels of an atom far away from the defect can be done according to

$$\Delta\varepsilon_F = (\varepsilon_F - \varepsilon_{\text{core}}^{\text{D}}) - (\varepsilon_{\text{VBM}}^{\text{perf}} - \varepsilon_{\text{core}}^{\text{perf}}).$$

Plot the atom projected density of states (output option `output atom_proj_dos` in `control.in`) for this atom for the charged defected and the neutral perfect system to visualize changes in the core states. (Be aware that in an all-electron approach the deeply lying core states are sensitive to local changes in electron density due to relaxation and charge redistribution, so that their shift in a defected system with respect to the perfect host system may not include only the average potential shift.)

Due to spurious electrostatic interaction as a result of the employed periodic boundary conditions the formation energy of a charged defect depends on the dimensions of the supercell. The formation energy scales as  $\Delta G_f^{\text{D}}(L) \approx a \frac{1}{L} + c$  for sufficiently large supercells [73]. For a simple cubic unit cell  $L$  corresponds to the supercell lattice constant and can take up integer multiples of the unit cell lattice constant  $L^{(0)}$ . For differently shaped unit cells with lattice constants  $L_1^{(0)}$ ,  $L_2^{(0)}$ ,  $L_3^{(0)}$  set for example  $L := L_1$  and build supercells  $L_1 = n \cdot L_1^{(0)}$ ,  $L_2 = n \cdot L_2^{(0)}$ ,  $L_3 = n \cdot L_3^{(0)}$  with integer  $n$ . The desired formation energy of a single defect in an infinite supercell  $\Delta G_f^{\text{D}}(L \rightarrow \infty)$  can then be obtained by extrapolation. Note, that the convergence of the extrapolated energy with respect to the supercell size should be tested carefully. Taking into account geometric relaxation can improve the convergence significantly. Alternatively postprocessing correction schemes that allow to remove the spurious interaction terms have been suggested in literature [73, 32].

## Chapter 5

# The AITRANSS package

The AITRANSS (*ab initio* transport simulations) package is a project under continuous development at the Institute of Nanotechnology of the Karlsruhe Institute of Technology (KIT), Germany, since 2002. In brief, when combined with FHI-aims, AITRANSS provides a post-processor module that enables, e.g., calculation of the electron transport characteristics of molecular junctions based on a Landauer formalism in a (non-equilibrium) Green's function formulation.

Currently, the version of the code accessible to FHI-aims users is limited to computation of the ballistic (Landauer-Büttiker) transmission function. According to current planning advanced options, e.g., out of equilibrium transport response, will be available in the future releases.

A discussion of the underlying physical formalism and details of the implementation are described in the reference [6]:

Andreas Arnold, Florian Weigend, and Ferdinand Evers, "Quantum chemistry calculations for molecules coupled to reservoirs: Formalism, implementation, and application to benzenedithiol." *J. Chem. Phys.* **126**, 174101 (2007).

Please, cite the above work together with FHI-aims publications, when using AITRANSS.

For questions and bug reports, contact Alexej Bagrets (Alexej.Bagrets@kit.edu).

## 5.1 Source code and supporting materials

The source code and supporting material of the AITRANSS-module for the FHI-aims package is placed in the subdirectory `aitranss/`. This directory contains subdirectories:

- `source/` : with the Fortran90 code and the example Makefile ;
- `tcontrol.script/` : contains a script `tcontrol.aims.x`, which is served to prepare a mandatory input file `tcontrol` for the transport-type calculation ;
- `electrodes.library/` : contains a library of representative gold (Au) clusters (xyz-files) which should be linked, via anchoring groups, to your molecular system to create an “extended molecule”: its electronic structure (Kohn-Sham molecular orbitals and energies) is a prerequisite to compute transport characteristics ;
- `examples/` : contains examples, with input and output files of the FHI-aims and AITRANSS; README files found in this subdirectory contain also short guidelines on how an input for a particular transport calculation has been created.

## 5.2 Compiling the AITRANSS module

Please, use a template of the Makefile found in the directory `source/`, and adjust variables referring to compiler (FC and LD), compiler's options (FLAGS) and a path to libraries (LIBS) at your computer system. A mandatory prerequisite to build the code is a Fortran 90/95 capable compiler and a compiled version of LAPACK and BLAS (for example, Intel's MKL). A binary (`aitranss.x`) built by the Makefile will go to the `bin/` directory of the FHI-aims.

In contrast to FHI-aims, the current release of AITRANSS is not yet based on MPI. However, you are encouraged to use a fortran compiler option(s), aka `"-openmp"` and `"-O2"` for Intel's `ifort`, to enable the auto-parallelizer to build a multithreaded code based on OpenMP directives.

According to our experience, a generated code can be safely executed in parallel within a single compute node with multiple processors, and with a significant gain in computation time.

We advise you as well to copy a script `tcontrol.aims.x` found in the directory `tcontrol.script/` to the directory `bin/` of the FHI-aims installation, and to make files in that directory accessible for the execution from a command line by adjusting your shell variable `PATH`.

## 5.3 How to set-up and run transport calculations

### 5.3.1 FHI-aims run: input and output

Having your molecule "at hands", use your favorite modeling and visualization tools/-software, and prepare an extended structure by linking the molecule via anchoring groups to two atomic clusters, representing parts of macroscopic *source* and *drain* electrodes. Consult the `electrodes.library/` directory, and use predefined Au clusters found there. A typical example of an "extended molecule", which you are requested to build, is shown in Fig. 5.1.

Include a line

```
output aitranss
```

into your `control.in` file. Furthermore, following settings are recommended for the self-consistent DFT calculation:

```
occupation_type    gaussian 0.01
mixer              pulay
  n_max_pulay      10
  charge_mix_param 0.2

sc_accuracy_rho    1E-4
sc_accuracy_eev    1E-2
sc_accuracy_etot   1E-6

relativistic zora scalar 1.0e-10
```

Invoke FHI-aims exploiting a cluster type (non-periodic) calculation. After the FHI-aims run is finished, you'll find in your directory three ASCII files: `basis-indices.out`, `omat.aims` and `mos.aims`. These files contain: some limited information on basis functions; overlap integrals; and data on Kohn-Sham molecular orbitals & energies of the "extended molecule", respectively. If spin channels of your system are not identical, `mos.aims` will be substituted by two other files called `alpha.aims` and `beta.aims`.

### 5.3.2 What to be aware of before running AITRANSS module

The AITRANSS module should be run from the same directory, where output files of FHI-aims are placed. A file `geometry.in` is mandatory and should also be there.

A file `control.in` is not used. Instead, another mandatory file for the transport calculation is `tcontrol`. Please, *always* use a script `tcontrol.aims.x` to create this file. Executing a script `tcontrol.aims.x` without arguments outputs a help information:

[...]

```
-----
"tcontrol.aims.x"  script creates a mandatory file "tcontrol"
                   which is required to run the "aitranss"
                   post-processing module after FHI-aims
-----
```

USAGE: tcontrol.aims.x [ -option <argument> ] ...

where options & arguments are:

```

                                     ! electrodes geometry:
-lsurc <atom1>                       three atoms which define an outermost
-lsurx <atom2>                       LEFT surface layer of the extended
-lsury <atom3>                       molecule

-rsurc <atom4>                       three atoms which define an outermost
-rsurx <atom5>                       RIGHT surface layer of the extended
-rsury <atom6>                       molecule

-nlayers <number>                   number of atomic layers coupled to
                                     reservoirs via a self-energy

                                     ! energy window, in Hartree [H], to
                                     ! output transmission function T(E) :
-ener  <E1[H]>                       initial energy point, E1
-estep <dE[H]>                       energy step, dE
-ehend <E2[H]>                       final energy point, E2

                                     ! output :
-outfile <file_name>                 output file name for T(E) [default: TE.dat]
```

When executed with options and arguments, a script `tcontrol.aims.x` checks for the `geometry.in` file and other mandatory FHI-aims output files (`basis-indices.out`, `omat.aims`, `mos.aims` or `alpha.aims` & `beta.aims`) in your directory, reads from these files information on a system size and the Hamiltonian  $H$  and overlap matrix dimension, and exports this information together with your arguments to an ASCII file `tcontrol`. Options and arguments are used: (i) to provide information on the self-energy construction; (ii) to introduce an energy window for the calculation of the transmission function  $T(E)$ , and (iii) (optionally) to introduce an output file name for  $T(E)$ .

*Comment on the self-energy.* When an “extended molecule” is contacted to macroscopic reservoirs, a propagation of an electron with energy  $E$  within a subspace limited by the “extended molecule” is described by the Green’s function:  $G^{-1}(E) = E - H - \Sigma(E)$ , where a self-energy  $\Sigma(E)$  accounts for the interaction between a finite system and macroscopic reservoirs. As argued in Refs. [6, 29], if atomic clusters introduced to model parts of metallic electrodes are large enough, the reservoirs can be modeled by

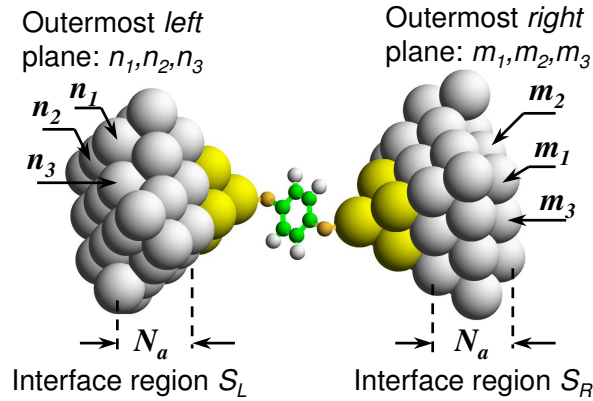


Figure 5.1: A schematic view of the “extended molecule”. The shaded regions are interfaces to the two reservoirs. Within the interface regions absorbing boundary conditions are active and the self-energy  $\Sigma$  is introduced. A user defines interface regions by specifying the outermost *left* and *right* atomic planes (introducing indices of three different atoms that form a triangle,  $n_1, n_2, n_3$ , and  $m_1, m_2, m_3$ , respectively) and the amount of atomic layers,  $N_a$ .

absorbing boundary conditions which become active at the interface regions  $S_L$  and  $S_R$  (labeled by a gray color in Fig. 5.1) where the “extended molecule” is coupled to reservoirs. Within this model, the self-energy is approximated by an energy-independent, diagonal matrix,

$$\Sigma_{nn'}^{\mu\mu'} \approx -i\eta_n \delta_{nn'} \delta_{\mu\mu'},$$

where indices  $n$  and  $n'$  label atoms, and  $\mu$  and  $\mu'$  label corresponding internal degrees of freedom (i.e., atom-centered basis functions). Here absorption rates  $\eta_n$  are allowed to have non-zero weights only within the interface regions  $S_L$  and  $S_R$ . A user defines interface regions by specifying the outermost *left* and *right* atomic planes (introducing indices of three different atoms forming a triangle,  $n_1, n_2, n_3$ , and  $m_1, m_2, m_3$ , respectively) and the amount of atomic layers,  $N_a$ .

### 5.3.3 How to create a mandatory file `tcontrol`

To create a `tcontrol` file, a `tcontrol.aims.x` script should be launched with the following options and arguments:

```
tcontrol.aims.x -lsurc  $n_1$  -lsurx  $n_2$  -lsury  $n_3$  -rsurc  $m_1$  -rsurx  $m_2$  -rsury  $m_3$  -nlayers  $N_a$  -ener  $E_1$  -estep  $dE$  -eend  $E_2$ 
```

- where integer numbers  $n_1, n_2, n_3$  are indices of three different atoms fixing the outermost *left* atomic plane of the “extended molecule” (see Fig. 5.1); atoms are numbered according to their appearance in file `geometry.in` ;
- integer numbers  $m_1, m_2, m_3$  are indices of three different atoms fixing the outermost *right* atomic plane of the “extended molecule” (see Fig. 5.1);

- an integer  $N_a$  indicates the number of atomic layers defining the interface regions  $\mathcal{S}_L$  and  $\mathcal{S}_R$  (see Fig. 5.1); users are strongly advised to take Au clusters of similar size from the directory `electrodes.library/` and use a parameter  $N_a$  suggested in the header of a library file. Using Au clusters of similar size insures that  $N_a$  can be consistently chosen to be the same for both *left* and *right* interface regions;
- real numbers  $E_1$ ,  $E_2$  and a positive real number  $dE$  define the energy window  $[E_1, E_2]$  and energy step  $dE$  to calculate transmission function  $T(E)$ ;
- optionally, you can launch the script with "`-output filename`" where *filename* is a string, specifying a name of the output file for  $T(E)$ .

An example of the transport calculation set-up for the Au-benzene-dithiol-Au junction can be found in the directory `examples/au-c6h6-au/`. A script `tcontrol.aims.x` has been executed there with the following options and arguments:

```
tcontrol.aims.x -lsurc 34 -lsurx 30 -lsury 36 -rsurc 35 -rsurx 33
               -rsury 31 -nlayers 2 -ener -0.4000 -estep 0.0001 -eend 0.0000
```

to create the following file `tcontrol`:

```
#input data for the aitranss module
$aims_input on
$landauer on
$coord   file=geometry.in
$natoms  50
$basis   file=basis-indices.out
$read_omat file=omat.aims
$scfmo   file=mos.aims
$nsaos   2268
$lsurc   34
$lsurx   30
$lsury   36
$rsurc   35
$rsurx   33
$rsury   31
$nlayers 2
$s1i     0.1d0
$s2i     0.05d0
$s3i     0.025d0
$ener    -0.4000
$estep   0.0001
$eend    0.0000
$output  file=TE.dat
$testing off
$end
```

Detailed description of keywords in file `tcontrol` are given in the paragraph 5.4.

### 5.3.4 How to submit a transport calculation and its output

Assuming a binary file `aitranss.x` is placed in the directory `bin/` of the FHI-aims installation which is referenced in your shell variable `PATH`, you can run the transport type calculation from a command line, like

```
> aitranss.x | tee my.transport.calc.out
```

or

```
> nohup aitranss.x > my.transport.calc.out &
```

FHI-aims output files together with information provided from files `tcontrol` and `geometry.in` will be used by the `AITRANSS` module to reconstruct a Kohn-Sham Hamiltonian ( $H$ ) of the “extended molecule”, to supplement  $H$  with the self-energy  $\Sigma$ , and to compute a ballistic (Landauer-Büttiker) transmission function, exploiting Green’s function formalism.

After calculation is finished, you’ll get two files. The first file with a default name `TE.dat` contains information on the transmission function,  $T(E)$ . Data in this file are arranged in columns as indicated in a file’s header: (i) first column is energy in Hartree (atomic) units; (ii) second column is energy in eV units given with respect to the Fermi energy, which is also stated in the file’s header; (iii) third column contains data on ballistic transmission per spin. If spin channels of your system are different, there will be present third and fourth columns, referring to transmission in up-spin ( $\alpha$ ) channel and down-spin ( $\beta$ ) channel, respectively. A contribution to conductance due to spin  $\sigma$  electrons is given by transmission at the Fermi energy,  $T_\sigma(E_F)$ , in units  $e^2/h$ .

The second file has a name `self.energy.in`, and contains information about the model self-energy construction. A format of this file is similar to the format of `geometry.in`, where each line corresponds to one atom in the structure, and atom’s specific data are arranged in columns: (i) columns from 1st till 5th are atom index  $n$ , its  $x$ ,  $y$  and  $z$ -coordinates (in Å), and atomic symbol, respectively; (ii) 6th column can contain entries `left`, `right` or `empty`, depending on whether a given atom  $n$  is a part of the left interface region  $\mathcal{S}_L$ , right interface region  $\mathcal{S}_R$ , or does not belong to none of them (see Fig. 5.1 for details); (iii) last column contains a real number in a “double precision” fortran-like format: this number is a local leakage rate  $\eta_n$ , in Hartree units, at given atom  $n$  (for details, please see a comment on the model self-energy construction, section 5.3.2).

Default leakage rates  $\eta_n$ ’s, used for construction of the model self-energy, have extensively been tested in numerous previous transport studies with Au-electrodes. These rates are referenced also in the `tcontrol` file and controlled by the keywords (tags) `$s1i`, `$s2i` and `$s3i` as explained in detail in the following paragraph 5.4.

*Warning: If you are not a transport-expert user and employ the `AITRANSS` module as a “black-box” for routine transport calculations, we advise you against modifying the default parameters. This may easily lead to misleading or even completely incorrect results.*

If the post-processor transport calculation is resubmitted, a previously created file `self.energy.in` will be used to initialize the self-energy matrix, as controlled by the keyword `$self_energy` in file `tcontrol`, see section 5.4.

## 5.4 Keywords of file `tcontrol`

All keywords (tags) of `tcontrol` file begin with a special symbol `$`. Lines starting from `#` are comment lines. All lines after a keyword `$end` are ignored.

---

**Tag:** `$aims_input` (`tcontrol`)

Usage: `$aims_input` on

Purpose: mandatory flag, sets up the FHI-aims like format of input/output files.

---

**Tag:** `$landauer` (`tcontrol`)

Usage: `$landauer` on

Purpose: mandatory flag; request for the transmission calculation.

---

**Tag:** `$coord` (`tcontrol`)

Usage: `$coord` `file=geo-filename`

Purpose: mandatory keyword; sets up a file name with atomic positions (in FHI-aims format); *geo-filename* is a text string without spacings, e.g., `geometry.in`

A specified file should be present in your directory. The whole string, `file=geo-filename`, should not contain any spacings.

---

**Tag:** `$natoms` (`tcontrol`)

Usage: `$natoms` `n`

Purpose: mandatory keyword, specifies number of atoms in the “extended molecule”; `n` is a positive integer number.

---

**Tag:** `$basis` (`tcontrol`)

Usage: `$basis` `file=basis-filename`

Purpose: mandatory keyword; sets up a file name with information on basis functions quantum numbers as written out by the FHI-aims; *basis-filename* is a text string without spacings, default file name is `basis-indices.out`

A specified file should be present in your directory. The string `file=basis-filename` should not contain spacings.

---

**Tag:** `$read_omat` (*tcontrol*)

Usage: `$read_omat file=overlap-filename`

Purpose: mandatory keyword, sets up a file name with overlap matrix elements as written out by the FHI-aims; *overlap-filename* is a text string without spacings, default file name is `omat.aims`.

A specified file should be present in your directory. The string `file=basis-filename` should not contain spacings.

---

**Tag:** `$scfmo` (*tcontrol*)

Usage: `$scfmo file=mos-filename`

Purpose: mandatory keyword in case of non-spin-polarized calculation; sets up a file name with self-consistent-field molecular orbitals (e.g., Kohn-Sham wave functions) as written out by the FHI-aims; *mos-filename* is a text string without spacings, default file name is `mos.aims`.

A specified file should be present in your directory. The string `file=mos-filename` should not contain spacings.

---

**Tags:** `$uhfmo_alpha` (*tcontrol*)

`$uhfmo_beta` (*tcontrol*)

Usage: `$uhfmo_alpha file=alpha-filename`

`$uhfmo_beta file=beta-filename`

Purpose: mandatory keywords in case of spin-polarized calculation; set up file names with self-consistent-field molecular orbitals (e.g., Kohn-Sham wave functions) as written out by the FHI-aims for  $\alpha$  (up-spin) and  $\beta$  (down-spin) electrons, respectively; *alpha-filename* and *beta-filename* are text strings without spacings, default file names are `alpha.aims` and `beta.aims`.

Specified files should be present in your directory. The strings `file=alpha-filename` and `file=beta-filename` should not contain spacings.

---

**Tag:** `$nsaos` (*tcontrol*)

Usage: `$nsaos N`

Purpose: mandatory keyword, specifies dimension *N* of the overlap matrix and a single-particle Hamiltonian of the “extended molecule”.

*N* is a positive integer number; its value can be found in the header line of default output FHI-aims files `omat.aims` and `mos.aims` (or `alpha.aims` and `beta.aims`).

**Tags:** `$lsurc` (tcontrol)  
`$lsurx` (tcontrol)  
`$lsury` (tcontrol)

Usage: `$lsurc`  $n_1$   
`$lsurx`  $n_2$   
`$lsury`  $n_3$

Purpose: mandatory keywords; integer numbers  $n_1$ ,  $n_2$  and  $n_3$  are indices of three different atoms according to their appearance in file `geometry.in`, which define in a unique way an outermost *left* atomic surface of the “extended molecule” (see Fig. 5.1 for details).

---

**Tags:** `$rsurc` (tcontrol)  
`$rsurx` (tcontrol)  
`$rsury` (tcontrol)

Usage: `$rsurc`  $m_1$   
`$rsurx`  $m_2$   
`$rsury`  $m_3$

Purpose: mandatory keywords; integer numbers  $m_1$ ,  $m_2$  and  $m_3$  are indices of three different atoms according to their appearance in file `geometry.in`, which define in a unique way an outermost *right* atomic surface of the “extended molecule” (see Fig. 5.1 for details).

---

**Tag:** `$nlayers` (tcontrol)

Usage: `$nlayers`  $N_a$

Purpose: integer number  $N_a$  specifies amount of atomic layers within interface regions at the boundaries of “extended molecule” where absorbing boundary conditions are active and self-energy matrix elements (leakage rates) are non-zeros, see Fig. 5.1 for details.

*An integer number  $N_a$  should be taken from a header line of library files for Au electrodes which are placed in `electrodes.library/` directory.*

When using `tcontrol.aims.x`, the number  $N_a$  is passed to a script by the option: `-nlayers  $N_a$` .

---

**Tags:** `$s1i` (tcontrol)  
`$s2i` (tcontrol)  
`$s3i` (tcontrol)

Usage: `$s1i`  $\eta_1$   
`$s2i`  $\eta_2$   
`$s3i`  $\eta_3$

Purpose: positive real numbers  $\eta_1$ ,  $\eta_2$  and  $\eta_3$  define local leakage rates (in Hartree units) which parametrize self-energy matrix elements.

Default values of leakage rates for Au clusters, written by the script `tcontrol.aims.x` to the file `tcontrol`, reflect a gradual switching of perturbation and are given by:  $\eta_1 = 0.1$  for the outermost atomic layer of the “extended molecule” (see Fig. 5.1 for details);  $\eta_2 = 0.05$  for the next-to-the-outermost atomic layer; and  $\eta_3 = 0.025$  for the rest of atomic layers within the interface regions of the “extended molecule”.

See also a keyword `$self_energy`.

*Disclaimer: 'Black-box'-users of AITRANSS are strongly advised to work with the Au-electrodes listed in the library and use default parameters for the self-energy, only. The transport code will print out a warning message if other chemical elements are employed as electrodes material. Unexpected modification of electrodes or self-energy settings will, in general, lead to misleading or incorrect scientific results.*

**Tags:** `$ener` (`tcontrol`)  
`$estep` (`tcontrol`)  
`$eend` (`tcontrol`)

Usage: `$ener`  $E_1$   
`$estep`  $dE$   
`$eend`  $E_2$

Purpose: real numbers  $E_1$ ,  $dE$  and  $E_2$  should be given in Hartree units and define the energy window  $[E_1, E_2]$ , and the energy step  $dE$  for calculation and output of the transmission function  $T(E)$ .

If any of above mentioned keywords is missing, only conductance at the Fermi energy is calculated.

**Tag:** `$self_energy` (`tcontrol`)

Usage: `$self_energy` `file=self-energy-file`

Purpose: sets up a name of the file with atom specific values parameterizing the self-energy matrix; a format of the self-energy file is explained in section 5.3.4.

`self-energy-file` is a text string, a default file name is `self.energy.in`. The string `file=self-energy-file` should not contain spacings.

If a keyword `$self_energy` is present in `tcontrol`, the diagonal elements of the self-energy matrix are read from the referenced file. In that case, parameters and values given by keywords `$lsurc`, `$lsurx`, `$lsury`, `$rsurc`, `$rsurx`, `$rsury`, `$nlayers`, `$s1i`, `$s2i`, and `$s3i` do not have an effect.

**Tag: \$testing** (tcontrol)Usage: `$testing` flag

Purpose: optional keyword, reserved for testing purposes; flag can take values on or off. Default value is set to off.

If flag is set to on, several internal checks are performed to insure that employed numerical procedures give correct results, e.g.: (i) eigenvalues of the reconstructed Kohn-Sham Hamiltonian  $H$  coincide with values stored in files `mos.aims`, `alpha.aims` or `beta.aims`, and eigenvectors of  $H$  are orthogonal to each other; (ii) eigenvalues of the overlap matrix are positive, and the square-root of the overlap matrix multiplied by itself gives back the overlap matrix; (iii) a matrix  $B$  of eigenvectors of the complex valued operator  $H + \Sigma$  multiplied by the inverse  $B^{-1}$  gives a unitary matrix,  $BB^{-1} = 1$ , etc. Furthermore, there appear many `.tmp` files: one of them called `zmos.tmp` (or `zalpha.tmp` and `zbeta.tmp`) contains information on the complex poles  $E_n = \varepsilon_n - i\delta_n$  of the Green's function  $G^{-1}(E) = E - H - \Sigma$ .

---

**Tag: \$end** (tcontrol)Usage: `$end`Purpose: mandatory keyword, indicates the last line of file `tcontrol`, which is read by the transport module AITRANSS. All lines below this one are ignored.

*Acknowledgment: A. Bagrets and F. Evers acknowledge the help of Richard Korytár in writing a manual on AITRANSS.*

# Appendix A

## Trouble-shooting

We sincerely hope that FHI-aims will largely “do the job” for you as it comes; in fact, a large amount of work has gone into ensuring sane responses and understandable output from code when something was requested that was not safe or reasonable to do. Nonetheless, as with every piece of software, non-trivial issues can happen that are not immediately obvious to the user. In this appendix, we provide a list of known conditions that have taken unwary users by surprise, how to detect and how to fix them.

If you know of an issue that is not discussed below, but that should be included because it presents an easy stumbling block especially for new/inexperienced users, please let us know, and we will address the issue here.

### A.1 Format flags required by some compilers

FHI-aims contains source code files in different formats (.f or .f90), and sometimes containing rather long lines in the .f90 versions.

The Makefile therefore contains two different versions of the compiler flags, FFLAGS and F90FLAGS, which can be the same for some compilers, but do not have to be the same.

For specific compilers, flags that must be added to account for the different file formats properly are:

- xlf90 compiler (IBM): FFLAGS must contain the option “-qfixed” in addition to all other options specified with the F90FLAGS.
- g95 compiler: F90FLAGS should contain the option -ffree-line-length-huge in addition to all other options found in FFLAGS.

## A.2 FHI-aims aborts with a segfault at the beginning of the first test run.

We here repeat the information given already in Chapter 1:

If you are not familiar with Unix or Unix-like operating systems, the following will perhaps clarify what is going on. In Unix, the operating system *kernel* will allow a program to allocate / deallocate the variables it requires on the so-called *heap*. For small quick variables needed at runtime, this is not always the most efficient procedure (you do not want to allocate / deallocate very single loop counter in your code, for example). Such small variables can instead be requested from a *stack*, available per process, and also controlled by the kernel.

In principle, using the stack is not a great problem on current computers available for scientific computing, because you will rarely ever find more than a few processes at the same time that make excessive use of the stack. So, technically it should not matter whether you get your memory from the heap, or from the stack.

Unfortunately, for reasons unbeknownst to us, some operating system vendors limit the default user stack size to  $\approx 5$  MB in a time when typical available RAM per processor is 2 GB or more. For some purposes, FHI-aims *requires* that the execution stack size available to you be large enough for some initial internal operations. If too little stack is available, your FHI-aims run will *segfault* shortly after the command was launched. In that case, type:

```
> ulimit -s unlimited
```

(when using the bash shell or similar), or

```
> limit stacksize unlimited
```

(when using the tcsh or similar).

```
> echo $SHELL
```

will tell you which shell you are using. Ideally, this same setting should be specified in your `.profile`, `.bashrc`, or `.cshrc` login profiles. If “unlimited” does not work, try setting a large value instead, e.g., `ulimit -s 500000`.

If any of these steps are not allowed, you will have to contact the system manager of your computer in order to modify the stack size limits set by the kernel of your operating system.

FHI-aims prints at startup the settings of the stacksize as they are found on your system. As mentioned, here “unlimited” or a large value should be reported.

## A.3 Use of FHI-aims with multithreaded BLAS (e.g., Intel's mkl)

The performance of FHI-aims depends critically on the basic linear algebra subroutine (BLAS) library used to perform matrix operations. Such libraries are highly CPU-specific, and should be provided and optimized by yourself or your computer vendor for your particular computer.

Unfortunately, with the advent of multi-core CPUs for PCs, some computer vendors (Intel, IBM) have decided that their proprietary BLAS implementations will, by default, use *all* available CPU's by way of *threads*, since they do not expect a user to know how to create parallel code.

In contrast, FHI-aims makes great efforts to distribute its workload evenly itself (much more efficient than leaving the task up to the BLAS, which are used for some, but by no means all operations in the code). Thus, FHI-aims invokes the correct number of sub-processes via the message-passing interface (MPI), then distributing any further basic numeric operations (matrix multiplications) using BLAS routines correctly itself.

If the default settings provided by a vendor are to use *all* CPUs for *every single* call or the BLAS operations, on a system with  $n$  CPUs this will lead to  $n \times n$  tasks running in parallel – not good at all for efficiency.

The problem is easily fixed by setting the system variable `OMP_NUM_THREADS` (number of threads invoked by OpenMP-parallelized libraries, e.g., BLAS) to 1:

```
export OMP_NUM_THREADS=1
```

This syntax is correct for the bash shell). When using Intel's mkl, you may likewise wish to set `MKL_NUM_THREADS` to 1. On top of this, Intel will still ignore your choice unless you set the less well documented variable `MKL_DYNAMIC` to `FALSE`.

Another, much simpler and equally well performing option is to use the freely available Goto BLAS subroutines that can be downloaded and compiled on standard architectures.

Some versions of Intel's mkl are known to have an error in a function called "pdtran". Thus at startup FHI-aims tests the version of pdtran it is currently using for correctness. Should FHI-aims abort with an error message "pdtran test failed! Aborting..." you will have to use a different version of Intel' mkl or replacements like Goto Blas.

## A.4 Parallel runs across different file systems

In parallel runs on distributed computers (clusters), FHI-aims expects its input files in the directory from which it is invoked. Its standard output can be redirected by hand to any given location, and other (optional, see keyword [output](#)) output files are again written in the same directory from which FHI-aims is invoked.

This procedure works well on most standard cluster and/or high-performance computing architecture available today, but you must make sure that the directories for input and output are visible and readable / writable on all the nodes across which FHI-aims is parallelized for a given run.

## A.5 Nearly singular basis sets: Strange results from small-unit-cell periodic calculation with many $k$ -points

We have observed numerous times that periodic bulk calculations with small unit cells and many  $k$ -points are apparently much more prone to ill-conditioning of the basis set than any other type of calculation. The symptom is that, with the usual accuracy and grid settings, large but still affordable basis sets (e.g., tier 3) will show reasonable convergence behavior at the outset, but then suddenly show a large jump and unphysical total energies at some point in the s.c.f. cycle.

The underlying reason is that the basis sets used by FHI-aims are overlapping and non-orthogonal. As the basis functions located at each atom of the structure approach completeness, the basis set as a whole becomes overcomplete. The result may be that certain linear combinations of basis functions are approximately expressible as linear combinations of some others. The eigenvalue problem Eq. (3.24) becomes *ill-conditioned*, and small amounts of numerical noise in the Hamiltonian / overlap matrix elements can group together to produce large unphysical effects in the eigenvalue spectrum.

If this happens, a number of strategies are available to deal with this situation. These are summarized in the following. Note, however, that ill-conditioning does indicate that your chosen basis set is already closer to completeness than even your computer can handle, and a smaller basis set for production calculations should be equally sufficient (and much faster) for high-quality results.

- Employ the `basis_threshold` keyword. This allows to identify the near-linear dependent components of the overlap matrix and eliminate them from the calculation. The successful threshold value depends on your chosen basis set and system, so test different choices (typically,  $10^{-4}$  or  $10^{-5}$ ). Note, however, that a large `basis_threshold` value may also impact the total energy found at a level of a few meV/atom.
- In addition, the keyword `override_illconditioning` must be set in order to run with a basis set that is reduced by `basis_threshold`. This should serve as an indicator that extra care is required in this situation—in particular, a detailed convergence analysis of the behaviour of the problem with increasing basis set size, and (separately!) with increasing cutoff radius, up to the value you are using. In most cases, it should turn out that either the basis set, or the cutoff radius, or both, were chosen to be far overconverged.
- Increase the accuracy of the integration grids via `radial_base` and `angular_grids`.

This is an expensive strategy (use for proof-of-principle only!), but it will serve to reduce the numerical noise in your calculations and thus increase the validity range of the eigenvalue solution, Eq. (3.24).

Again, note that we do not usually observe any ill-conditioning related problems for large periodic structures (e.g., surface slabs) or even very large molecules, even when employing very large basis sets.

## A.6 No convergence of the s.c.f. cycle even after many iterations

Successful strategies for s.c.f. convergence in standard electronic structure problems have been developed in the field for a long time. Still, there remain some particular pathological classes of calculations, and even some of particular physical interest: large metallic slabs, where charge oscillations can occur; spin-polarized systems with closely competing spin states; systems near the crossing of two Kohn-Sham eigenvalues at the Fermi level; etc.

The standard s.c.f. convergence strategy within FHI-aims is to use `mixer pulay`, with a configurable `charge_mix_param` and number of mixed iterations `n_max_pulay`. These, possibly together with a `preconditioner`, should be modified first in order to see whether the problem can be contained.

Beyond this, s.c.f. convergence issues can be highly system-specific in our experience, and general guidelines are hard to give. Things that will always work to some extent are:

- a linear `mixer` with a (very!!) small `charge_mix_param`, which in the limit will guarantee convergence, albeit at the expense of excessively many s.c.f. cycles to reach convergence
- A increased broadening specified with `occupation_type`. This is essential especially for metallic systems, but for small clusters, the quality of the obtained total energies will deteriorate somewhat as a result, since these suddenly correspond to fractional (very high temperature) occupation numbers around the Fermi level.

For particularly hardy cases, we strongly recommend to review in detail all the options available in Sec. 3.10.

While trying out all these options, however, we strongly suggest to use the `output_level full` keyword, in order to have the actual eigenvalue spectrum printed across different s.c.f. iterations, and then to *visualize* the behaviour of the eigenvalue spectrum as a function of s.c.f. iteration. In many cases, this step may yield some critical physical insight into the nature of the problem. For example, Kohn-Sham density functional theory may sometimes be forced to place competing electronic levels (*d* and *f* in rare earth elements are a good example, but there are many others!) at the Fermi level in order to ensure a given (ground-state) fractional occupation. The search for the correct occupation of these levels will then oscillate between different iterations, and could be

the source of the instability. Stabilizing such a problem is still not easy, but at least, looking at the electronic structure as it develops may give some critical hint as to what is happening, instead of leaving the user groping in the dark.

## Appendix B

### Structure of the code

The bulk of this manual is concerned with the available options to build and run FHI-aims for a particular purpose. In our experience, most users will probably remain at this level in their use of the code, already due to the time constraints of a normal research schedule.

Nonetheless, we strongly believe that running a “physics” code as a complete black-box package is not a good idea. Our branch of physics is based on (mostly) well-understood differential equations, and the solution technique applied, including its limitations, must be well understood, or at least understandable, to gauge the outcome of a particular calculation. In order to achieve this, the source code to the method used must be available. While we do not expect most users to go through the code in its entirety with a fine-toothed comb, we *do* encourage any user to look into the source code and try to understand in which way FHI-aims produces its exact solution to a problem.

The purpose of this appendix is to facilitate this understanding, by outlining the overall structure of the code, specifically, the high-level subdivision of physical tasks.

#### B.1 Flow of the program

The uppermost level of FHI-aims is the subroutine `main.f90`, the structure of which is shown as a structogram in Fig. B.1. As a subroutine, `main.f90` can also be called by external code as a library subroutine, with the restriction that, for parallel execution, this can happen only once [by definition of the message passing interface (MPI) for parallel communication, there can only be one call to `mpi_init` per program run].

The purpose of `main.f90` is to provide a logical separation of groups of computational tasks by way of high-level wrapper subroutines (listed in typewriter font in Fig. B.1). With the exception of global convergence checks, no outright physical quantities are directly manipulated in `main.f90`. All physically relevant quantities are handled inside the lower-level structure of the code and, if necessary, are passed between them by way of specific modules. For example, the module `physics.f90` handles all variables of tangible physical importance (Hamiltonian and overlap matrices, Kohn-Sham wave function, electron densities, potentials, ...). The geometry information for a given electronic

Initial timings and MPI setup <code>initialize_mpi, initialize_timings, initial_mpi_report</code>	
Read input data: Parse control.in, geometry.in for initial dimensions, allocate input data structures, read full content of control.in, geometry.in and verify consistency <code>read_input_data</code>	
Prepare data for all s.c.f. cycles: per-species integration grids, spherical free-atom DFT potentials and densities, radial basis functions $u(r)$ for each species <code>prepare_scf</code>	
Initial s.c.f. iteration: Partition and prepare 3-d integration grid, obtain overlap matrix, initial superposition-of-free-atoms Hamiltonian, Kohn-Sham eigenvalues and eigenvectors <code>initialize_scf</code>	
Full s.c.f. cycle - starting from initial Kohn-Sham eigenvectors, obtain self-consistent potential electron density, wave function, total energy and forces; track wall-time, s.c.f. convergence <code>scf_solver</code>	
No	Geometry optimization or molecular dynamics requested?
	Yes
	Predict next geometry step; check validity of forces; check geometry convergence <code>predict_new_geometry</code>
	While (enough walltime left) and (geometry not converged) and (valid forces)
	Repartition 3-d integration grids, recompute overlap matrix and fixed sums of free atoms for updated geometry: <code>reinitialize_scf</code> Full s.c.f. cycle - starting from previous Kohn-Sham eigenvectors, obtain self-consistent potential, electron density, wave function, total energy and forces; track wall-time, s.c.f. convergence <code>scf_solver</code> Predict next geometry step; check validity of forces; check geometry convergence <code>predict_new_geometry</code>
If requested, post-processing of wave function and density: Electrostatic moments, Mulliken & Hirshfeld charge analyses, volumetric (cube) output, MP2 perturbative correlation energy, GW or MP2 self-energy corrections <code>output_dipole_moment, output_quadrupole_moment, mulliken_analysis, hirshfeld_analysis, output_cube_files, prepare_corr_energy_calc, qpe_calculation, mp2_calculation</code>	
Final tasks: Deallocations, final timing output, finalize MPI infrastructure <code>final_deallocations, finalize_scalapack, final_timings, finalize_mpi</code>	

Figure B.1: High-level program flow of FHI-aims

structure cycle (coordinates) are found in module `geometry.f90`; etc.

More details regarding these and other modules are included with the code distribution as a separate document. The point here is that `main.f90` should never need to use any but the highest-level modules explicitly, i.e.:

- `dimensions.f90` : Inclusion of array dimensions for consistent allocations across the code, and wrapper flags to prevent access to unallocated variables which are not needed for a given task.
- `localorb_io.f90` : Wrapper module for consistent writing of output in parallel runs
- `mpi_utilities.f90` : Wrapper module for MPI initialization, finalization (first and last tasks of a run, respectively), and task distribution
- `timing.f90` : Wrapper module including all timing and accounting information, including the count of s.c.f. iterations, relaxation or MD steps.

The first task of `main.f90` is to initialize any accounting (timing etc.) information, the infrastructure required for MPI (or, to silently switch off the use of MPI entirely in the case of non-parallel runs), and to record all this information (initial time stamps, code version, number of parallel tasks and computer system layout) in the standard output.

The obvious next task is to read and process all input information given in `control.in` and `geometry.in`. Internally, this is handled in three steps (see the wrapper subroutine `read_input_data.f90`): First, both input files are parsed once, while extracting only the dimension information needed to set up any necessary arrays / array dimensions needed to house the following input data. Organizing this information is the task of module `dimensions.f90`. Next, the information in `control.in` is read and checked for consistency, using `read_control.f90` for all general information, and repeated calls to `read_species_data.f90` for all `species`-related information. Finally, subroutine `read_geo.f90` reads the input data of `geometry.in`, and verifies its consistency with the data contained in `control.in`.

At the end of this step (subroutine `read_input_data.f90`), *all* input data from all input files should have been read and processed. It is important that any known conflicts, or incomplete settings, should have been verified at this stage, stopping the code with an error message if outrightly conflicting input information is detected. For completeness, we mention that any technical input settings of global interest (e.g., the handling of spin, relativity, or exchange-correlation) are collected and accessible through the top-level module `runtime_choices.f90`.

The following steps are the “household” steps of electronic structure theory.

Wrapper subroutine `prepare_scf.f90` sets up all structure-independent, fixed pieces of the calculation, and stores them for easy access in the actual self-consistency cycle. This includes all free-atom quantities (densities and potentials for the initialization), radial basis functions for all `species`, one-dimensional logarithmic and three-dimensional radial and angular integration grids, and fixed coefficients for the analytic long-range part of the Hartree potential.

Wrapper subroutine `initialize_scf.f90` performs the initial s.c.f. cycle of the electronic structure calculation. In this step, the full three-dimensional integration grid is filled with fixed initial quantities (superposition of free-atom densities, potentials, and partition functions), the overlap and Hamiltonian matrix integrals are performed for the first time, and the initial Hamiltonian and overlap matrices are used to determine the storage requirements in the event of sparse matrix storage. If a two-electron Coulomb operator is needed (hybrid functionals, Hartree-Fock, MP2, *GW* etc.), the three-center overlap matrix elements  $(ij|\mu)$  of Eq. (3.34) (see Sec. 3.19 for details) and the Coulomb matrix of the auxiliary “resolution of the identity” basis set [denoted  $V_{\mu\nu}$  in Eq. (3.34)] are precomputed. The most important task in `initialize_scf.f90` is the initial solution of the Kohn-Sham equations Eq. (3.24), providing a first solution of the wave function coefficients  $c_{jl}$ . These are the starting point of every iteration of the s.c.f. cycle in the following step, subroutine `scf_solver.f90`.

With all preliminary information available, the task of `scf_solver.f90` is to produce a self-consistent electron density, wave function, and all associated observables for a given, fixed nuclear geometry. The order of the cycle until convergence is reached is:

1. Calculation of the Kohn-Sham electron density associated with the current wave function,  $c_{jl}$
2. electron density mixing and preconditioning, to produce the *input* electron density for the next set of Kohn-Sham equations
3. decomposition of the electron density into atom-centered multipole fragments  $\delta\tilde{n}_{\text{at},lm}(\mathbf{r})$  [see Eq. (Eq;mp)], and construction of the multipole components of the Hartree potential,  $\delta\tilde{v}_{\text{at},lm}(\mathbf{r})$
4. construction of the full electrostatic potential  $v_{\text{es}}(\mathbf{r})$  on all points  $\mathbf{r}$  of the three-dimension integration grid
5. Integration of the updated Hamiltonian matrix elements,  $h_{ij}$
6. possible addition of two-electron exchange matrix elements to  $h_{ij}$
7. solution of the Kohn-Sham equations Eq. (3.24), to produce an updated wave function  $c_{jl}$
8. computation of updated total energy components, and check of all convergence criteria.

After convergence is reached, `scf_solver.f90` also performs some inevitable post-processing steps, including “scaled ZORA” perturbative corrections for the appropriate relativistic treatment, and band structure and density of states data output.

With a converged self-consistent solution at hand, the code can now perform any number of similar calculations for updated geometries, e.g., for a geometry optimization, molecular dynamics, etc. If so, an updated geometry is first produced by subroutine `predict_new_geometry.f90`. Note that this simple subroutine should also serve as the starting point for any other calculations involving multiple geometries, such as the

calculation of “serial” geometries along a given set of coordinates, etc. For the updated geometry, all geometry-related storage arrays in the calculation and the overlap matrix must be recomputed in subroutine `reinitialize_scf.f90`. Following this, subroutine `scf_solver.f90` is invoked again, and a new self-consistent solution is obtained.

The final step of the code is to produce, by post-processing, any information that can be obtained from the converged self-consistent wave function or electron density, including electrostatic moments, charge analyses, etc. Beyond this, only necessary cleanup tasks follow, most notably the deallocation of all storage arrays and the MPI infrastructure, and the final time accounting information.

## B.2 Commenting and style requests

Generally, no *strong* style conventions are enforced within FHI-aims, recognizing the fact that most programmers follow their own style conventions and preferences when it comes to details. There are, however, some conventions that should be followed in order to keep the code as a whole legible, and maintainable. When writing additional code, please adhere to these, using existing modules or subroutines as models where appropriate.

1. Please comment your work. Any necessary functionality should be accompanied with comments, enough so that *at least* someone familiar with the underlying mathematics can follow your code.
2. Please provide regular comment headers for your work. Every module and subroutine comes with comment headers in the style used by the *Robodoc* code management system (see, e.g., <http://www.xs4all.nl/~rfsber/Robo> for details and a manual). Beyond the possible use of Robodoc, we follow this convention because it provides a clear and unambiguous laundry list of items that are needed in any subroutine header: a description of the subroutine *purpose*, possibly its *input* and *output* data, and most importantly a *copyright* statement that is needed for every file in the code distribution.
3. Please keep your usage of Fortran conservative. Some ambitious constructs, while desirable when following formal techniques such as fully object-oriented programming, can still expose compiler bugs when too “un-Fortran-like” syntax is used.
4. In particular, compiler-dependent bugs are the reason why the use of pointers is strongly discouraged in FHI-aims. Even when following the textbook, allocating and deallocating pointers works differently with different compilers, opening the possibility of unexpected memory leaks outside our control. Please don’t do it — this problem *has* bitten us before.

## Bibliography

- [1] C. Adamo and V. Barone. *J. Chem. Phys.*, 110:6158, 1999. [43](#)
- [2] C. Van Alsenoy. *J. Comput. Chem.*, 9:620, 1988. [43](#), [159](#)
- [3] H.C. Andersen. *J. Chem. Phys.*, 72:2384, 1980. [128](#)
- [4] V. I. Anisimov, editor. *Strong Coulomb correlations in electronic structure calculations*. Gordon and Breach, New York, 2000. [35](#)
- [5] R. Armiento and Ann E. Mattsson. *Phys. Rev. B*, **72**:085108, 2005. [41](#)
- [6] A. Arnold, F. Weigend, and F. Evers. Quantum chemistry calculations for molecules coupled to reservoirs: Formalism, implementation, and application to benzenedithiol. *J. Chem. Phys.*, 126:174101, 2007. [249](#), [252](#)
- [7] J. Baker, J. Andzelm, A. Scheiner, and B. Delley. *J. Chem. Phys.*, 101:8894, 1994. [65](#)
- [8] A. Barducci, G. Bussi, and M. Parrinello. Well-tempered metadynamics: A smoothly converging and tunable free-energy method. *Phys. Rev. Lett.*, 100:020603, 2008. [242](#)
- [9] A.D. Becke. *J. Chem. Phys.*, 88:1053, 1988. [41](#)
- [10] J. Behler, B. Delley, S. Lorenz, K. Reuter, and M. Scheffler. *Phys. Rev. Lett.*, **94**:036104, 2005. [136](#)
- [11] J. Behler, B. Delley, K. Reuter, and M. Scheffler. *Phys. Rev. B*, **75**:115409, 2007. [136](#)
- [12] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler. *Ab initio* molecular simulations with numeric atom-centered orbitals. *Comp. Phys. Comm.*, **180**:2175, 2009. [45](#), [57](#), [60](#), [68](#), [69](#), [70](#), [78](#), [79](#), [83](#), [92](#), [94](#), [97](#), [100](#), [102](#), [103](#), [105](#), [117](#), [119](#), [122](#), [193](#), [198](#), [202](#), [214](#)
- [13] S.D. Bond, B.J. Leimkuhler, and B.B. Laird. The Nose-Poincare method for constant temperature molecular dynamics. *J. Comp. Phys.*, 151(1):114–134, 1999. [122](#)
- [14] S.F. Boys and I. Shavitt. *University of Wisconsin Rept.*, WIS-AF-13, 1959. [43](#), [159](#)

- [15] G. Bussi, D. Donadio, and M. Parrinello. Canonical sampling through velocity rescaling. *J. Chem. Phys.*, 126:014101, 2007. [129](#)
- [16] D. M. Ceperley and B. J. Alder. Ground state of the electron gas by a stochastic method. *Phys. Rev. Lett.*, **45**:566–569, 1980. [41](#)
- [17] Michele Ceriotti, Giovanni Bussi, and Michele Parrinello. Langevin equation with colored noise for constant-temperature molecular dynamics simulations. *Phys. Rev. Lett.*, 102:020601, Jan 2009. [129](#), [130](#)
- [18] Michele Ceriotti, Giovanni Bussi, and Michele Parrinello. Colored-noise thermostats *À la carte*. *Journal of Chemical Theory and Computation*, 6(4):1170–1180, 2010. [129](#), [130](#)
- [19] Michele Ceriotti, Michele Parrinello, Thomas E. Markland, and David E. Manolopoulos. Efficient stochastic thermostating of path integral molecular dynamics. *The Journal of Chemical Physics*, 133(12):124104, 2010. [129](#), [130](#)
- [20] D.J. Chadi and M.L. Cohen. *Phys. Rev. B*, **8**:5747, 1973. [48](#)
- [21] G. Crooks. Nonequilibrium measurements of free energy differences for microscopically reversible markovian systems. *J. Stat. Phys.*, 90:1481, 1998. [242](#)
- [22] B. Delley. *J. Chem. Phys.*, 92:508, 1990. [60](#)
- [23] B. Delley. *J. Comp Chem.*, 17:1152, 1995. [58](#)
- [24] M. Dion, H. Rydberg, E. Schröder, D. C. Langreth, , and B. I. Lundqvist. *Phys. Rev. Lett.*, 92:246401, 2004. [39](#), [40](#), [43](#), [151](#), [152](#), [156](#)
- [25] R. M. Dreizler and E. K. U. Gross. *Density Functional Theory*. Springer, Berlin, 1990. [5](#)
- [26] Brett I. Dunlap, Notker Rösch, and S. B. Trickey. Variational fitting methods for electronic structure calculations. *Mol. Phys.*, 108(21):3167, 2010. [163](#)
- [27] C. Eckart. *Phys. Rev.*, **47**:552, 1935. [113](#), [115](#)
- [28] K. Eichkorn, O. Treutler, H. Öhm, M. Häser, and R. Ahlrichs. *Chem. Phys. Lett.*, 240:283, 1995. [43](#), [159](#)
- [29] F. Evers and A. Arnold. Molecular conductance from ab initio calculations: Self energies and absorbing boundary conditions. *arXiv:cond-mat/0611401v1*, 2006. [252](#)
- [30] E. Fabiano, L. A. Constantin, and F. Della Sala. Generalized gradient approximation bridging the rapidly and slowly varying density regimes: A pbe-like functional for hybrid interfaces. *Phys. Rev. B*, **82**:113104, 2010. [41](#)
- [31] D. Frenkel and B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, second edition, 2002. [122](#)

- [32] Christoph Freysoldt, Jörg Neugebauer, and Chris G. Van de Walle. Fully ab initio Finite-Size corrections for Charged-Defect supercell calculations. *Phys. Rev. Lett.*, 102:016402, 2009. [248](#)
- [33] C.-L. Fu and K.-M. Ho. *Phys. Rev. B*, **28**:5480, 1983. [92](#)
- [34] Martin Fuchs and Matthias Scheffler. Ab initio pseudopotentials for electronic structure calculations of poly-atomic systems using density-functional theory. *Computer Physics Communications*, 119(1):67 – 98, 1999. [144](#), [145](#), [146](#)
- [35] M.J. Gillan. *J. Phys.:Condens. Matter*, 1:689, 1989. [93](#)
- [36] B Grabowski, L Ismer, T Hickel, and J Neugebauer. *Phys. Rev. B*, 79:134106, 2009. [132](#)
- [37] S. Grimme. *J. Chem. Phys.*, **20**:9095, 2003. [39](#), [43](#)
- [38] Andreas Grüneis, Martijn Marsman, Judith Harl, Laurids Schimka, and Georg Kresse. Making the random phase approximation to electronic correlation accurate. *J. Chem. Phys.*, 131:154115, 2009. [34](#)
- [39] A. Gulans, M. Puska, and R. Nieminen. Linear-scaling self-consistent implementation of the van der waals density functional. *Phys. Rev. B*, 79:201105(R), 2009. [156](#)
- [40] A. J. S. Hamilton. Uncorrelated modes of the non-linear power spectrum. *Mon. Not. R. Astron. Soc.*, 312(2):257–284, 2000. [167](#)
- [41] Andrew J. S. Hamilton. Fftlog, 2000. [167](#)
- [42] B. Hammer, L.B. Hansen, and J.K. Nørskov. *Phys. Rev. B*, 59:7413, 1999. [42](#)
- [43] Myung Joon Han, Taisuke Ozaki, and Jaejun Yu. O(N) LDA+U electronic structure calculation method based on the nonorthogonal pseudoatomic orbital basis. *Physical Review B*, 73(4):045110, January 2006. [44](#)
- [44] V. Havu, V. Blum, P. Havu, and M. Scheffler. *J. Comput. Phys.*, **228**:8367, 2009. [57](#), [58](#), [59](#), [61](#), [202](#)
- [45] L. Hedin. *Phys. Rev.*, 139:A796, 1965. [43](#)
- [46] G. Henkelman, B. P. Uberuagga, and H. Jonsson. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *J. Chem. Phys.*, **113**:9901, 2000. [241](#)
- [47] John M. Herbert and Martin Head-Gordon. Accelerated, energy-conserving Born-Oppenheimer molecular dynamics via fock matrix extrapolation. *Phys. Chem. Chem. Phys.*, 7:3269–3275, 2005. [123](#)
- [48] Jochen Heyd, Gustavo E. Scuseria, and Matthias Ernzerhof. *J. Chem. Phys.*, 118:8207, 2003. [42](#)

- [49] Jochen Heyd, Gustavo E. Scuseria, and Matthias Ernzerhof. *J. Chem. Phys.*, 124:219906, 2006. [42](#)
- [50] F.L. Hirshfeld. *Theor. Chim. Acta (Berl.)*, 44:129, 1977. [194](#)
- [51] P. Hohenberg and W. Kohn. *Phys. Rev. B*, 136:864, 1964. [5](#)
- [52] H. Ishida, Y. Nagai, and A. Kidera. *Chem. Phys. Lett.*, 282(2):115, 1998. [128](#)
- [53] A. Itoh and H. Matsunami. Single crystal growth of sic and electronic devices. *Critical Reviews in Solid State and Materials Sciences*, 22(2):111–197, 1997. [219](#)
- [54] C. Jarzynski. Nonequilibrium equality for free energy differences. *Phys. Rev. Lett*, 78:2690, 1997. [242](#)
- [55] Jmol. An open-source java viewer for chemical structures in 3D. <http://www.jmol.org/>. [224](#), [225](#)
- [56] J. Junquera, O. Paz, D. Sanchez-Portal, and E. Artacho. *Phys. Rev. B*, 64:235111, 2001. [52](#)
- [57] G.P. Kerker. *Phys. Rev. B*, **23**:3082, 1981. [102](#)
- [58] L. Kleinman and D. M. Bylander. Efficacious form for model pseudopotentials. *Phys. Rev. Lett.*, 48:1425–1428, May 1982. [144](#)
- [59] W. Kohn and L.J. Sham. *Phys. Rev.*, 140:A1133, 1965. [5](#)
- [60] Jiří Kolafa. Numerical integration of equations of motion with a Self-Consistent field given by an implicit equation. *Mol. Simul.*, 18:193, 1996. [123](#)
- [61] G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Comp. Mat. Sci.*, **6**:15–50, 1996. [92](#), [93](#)
- [62] Aliaksandr V. Krukau, Oleg A. Vydrov, Artur F. Izmaylov, and Gustavo E. Scuseria. *J. Chem. Phys.*, 125:224106, 2006. [42](#)
- [63] Thomas D. Kühne, Matthias Krack, Fawzi R. Mohamed, and Michele Parrinello. Efficient and accurate Car-Parrinello-like approach to Born-Oppenheimer molecular dynamics. *Phys. Rev. Lett.*, 98:066401, 2007. [122](#)
- [64] S. Kumar, J. M. Rosenberg, D. Bouzida, R. H. Swendsen, and P. A. Kollman. Multidimensional free-energy calculations using the weighted histogram analysis method. *J. Comput. Chem.*, 16:1339, 1995. [242](#)
- [65] A. Laio and M. Parrinello. Escaping free energy minima. *Proc. Natl. Acad. Sci.*, 20:12562, 2002. [242](#)
- [66] V.I. Lebedev. *Zh. Vychisl. Mat. mat. Fiz.*, 15:48, 1975. [58](#)
- [67] V.I. Lebedev. *Zh. Vychisl. Mat. mat. Fiz.*, 16:293, 1976. [58](#)

- [68] V.I. Lebedev and D.N. Laikov. *Doklady Mathematics*, 59:477, 1999. [58](#)
- [69] C.L. Lee, W. Yang, and R.G. Parr. *Phys. Rev. B*, 37:785, 1988. [41](#)
- [70] R. Lindh, A. Bernhardsson, G. Karlström, and P.-Å. Malmqvist. *Chem. Phys. Lett.*, 241:423, 1995. [112](#), [116](#), [119](#), [209](#)
- [71] P.O. Loewdin. *J. Chem. Phys.*, 19:1936–1401, 1951. [81](#)
- [72] Steven G. Louie, Sverre Froyen, and Marvin L. Cohen. Nonlinear ionic pseudopotentials in spin-density-functional calculations. *Phys. Rev. B*, 26:1738–1742, Aug 1982. [146](#)
- [73] G. Makov and M. C. Payne. *Phys. Rev. B*, 51:4014, 1995. [248](#)
- [74] M. Manninen, R. Nieminen, and P. Hautojärvi. *Phys. Rev. B*, 12:4012, 1975. [102](#)
- [75] N.D. Mermin. *Phys. Rev.*, 137:A1441, 1965. [93](#), [132](#)
- [76] A. Messiah. *Quantum Mechanics*. North-Holland Publishing Company Amsterdam, 1970. [80](#)
- [77] M. Methfessel and A. T. Paxton. *Phys. Rev. B*, **40**:3616, 1989. [92](#)
- [78] H.J. Monkhorst and J.D. Pack. *Phys. Rev. B*, **13**:5188, 1976. [47](#), [217](#), [218](#)
- [79] R.S. Mulliken. *J. Chem. Phys.*, 23:1833, 1955. [196](#)
- [80] D. Nabok, P. Puschnig, and C. Ambrosch-Draxl. *Phys. Rev. B*, 77:245316, 2008. [151](#)
- [81] R. Nieminen. *J. Phys. F*, 7:375, 1977. [102](#)
- [82] Anders M. N. Niklasson, C. J. Tymczak, and Matt Challacombe. Time-Reversible Born-Oppenheimer molecular dynamics. *Phys. Rev. Lett.*, 97:123001, 2006. [127](#)
- [83] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 2. edition, 2006. [119](#)
- [84] Aliaksandr V. Krukau Oleg A. Vydrov, Jochen Heyd and Gustavo E. Scuseria. *J. Chem. Phys.*, 125:074106, 2006. [42](#)
- [85] K. Parlinski, Z. Q. Li, and Y. Kawazoe. First-principles determination of the soft mode in cubic zro<sub>2</sub>. *Phys. Rev. Lett.*, **78**:4063–4066, 1997. [231](#)
- [86] J. P. Perdew, K. Burke, and M. Ernzerhof. Generalized gradient approximation made simple. *Phys. Rev. Lett.*, **77**:3865–3868, 1997. [41](#)
- [87] J. P. Perdew and Y. Wang. Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B*, **45**:13244–13249, 1992. [41](#)
- [88] J. P. Perdew and A. Zunger. Self-interaction correction to density-functional approximations for many-electron systems. *Phys. Rev. B*, **23**:5048–5079, 1981. [41](#)

- [89] J.P. Perdew, A. Ruzsinszky, G.I. Csonka, O.A. Vydrov, G.E. Scuseria, L.A. Constantin, X. Zhou, and K. Burke. *Phys. Rev. Lett.*, 100:136406, 2008. [42](#), [43](#)
- [90] A. G. Petukhov, I. I. Mazin, L. Chioncel, and A. I. Lichtenstein. Correlated metals and the LDA+U method. *Physical Review B*, 67(15):153106, April 2003. [36](#), [37](#)
- [91] William K. Press, Saul A. Teukolsky, William T. Vetterlin, and Brian T. Flannery. *Numerical Recipes*. Cambridge University Press, 3 edition, 2007. [119](#)
- [92] P. Pulay. Convergence acceleration of iterative sequences. the case of scf iteration. *Chem. Phys. Lett.*, **73**:393–398, 1980. [100](#)
- [93] Peter Pulay and Géza Fogarasi. Fock matrix dynamics. *Chem. Phys. Lett.*, 386:272–278, 2004. [123](#)
- [94] P. Pyykkö. *Chem. Rev.*, 88:563, 1988. [214](#)
- [95] X. Ren, P. Rinke, V. Blum, J. Wieferink, A. Tkatchenko, A. Sanfilippo, K. Reuter, and M. Scheffler. Resolution-of-identity approach to Hartree-Fock, hybrid density functionals, RPA, MP2 and GW with numeric atom-centered orbital basis functions. *New J. Phys.*, **14**:053020, 2012. [34](#)
- [96] X. Ren, P. Rinke, G. E. Scuseria, and M. Scheffler. in preparation. [34](#)
- [97] X. Ren, A. Tkatchenko, P. Rinke, and M. Scheffler. Beyond the random-phase approximation for the electron correlation energy: The importance of single excitations. *Phys. Rev. Lett.*, **106**:153003, 2011. [34](#)
- [98] H. N. Rojas, R. W. Godby, and R. J. Needs. *Phys. Rev. Lett.*, **74**:1827, 1995. [160](#)
- [99] B. Roux. The calculation of the potential of mean force using computer-simulations. *Comput. Phys. Comm.*, 91:275, 1995. [242](#)
- [100] A. Sayvetz. *J. Phys. Chem.*, **7**:383, 1939. [113](#), [115](#)
- [101] G.E. Scuseria and V.N. Staroverov. Progress in the development of exchange-correlation functionals. In C.E. Dykstra, G. Frenking, K.S. Kim, and G.E. Scuseria, editors, *Theory and Applications of Computational Chemistry: The First 40 Years*, chapter 24. Elsevier, Amsterdam, 2005. The parameters for "VWN"-LDA as implemented in the Gaussian code are given in Table 1. [41](#), [42](#)
- [102] P. Sherwood, A. H. de Vries, M. F. Guest, G. Schreckenbach, R. A. Catlow, S. A. French, A. A. Sokol, S. T. Bromley, W. Thiel, A. J. Turner, S. Billeter, F. Terstegen, S. Thiel, J. Kendrick, S. C. Rogers, J. Casci, M. Watson, F. King, E. Karlsen, M. Sjøvoll, A. Fahmi, A. Schäfer, and C. Lennartz. *J. Mol. Struct (Theochem)*, **632**:1, 2003. [143](#)
- [103] R.E. Stratmann, G.E. Scuseria, and M.J. Frisch. *Chem. Phys. Lett.*, 257:213, 1996. [60](#), [61](#)
- [104] James D. Talman. Numerical methods for multicenter integrals for numerically defined basis functions applied in molecular calculations. *Internat. J. Quant. Chem.*, 93(2):72–90, 2003. [167](#)

- [105] J.D. Talman. NumSBT: a subroutine for calculating spherical bessel transforms numerically. *Comput. Phys. Comm.*, 180(2):332–338, February 2009. [167](#)
- [106] A. Tkatchenko and M. Scheffler. *Phys. Rev. Lett.*, 102:073005, 2009. [148](#), [150](#)
- [107] G. Torrie and J. Valleau. Nonphysical sampling distributions in monte carlo free energy estimation: Umbrella sampling. *J. Comput. Phys.*, 23:187, 1977. [242](#)
- [108] O. Vahtras, J. Almlöf, and M.W. Feyereisen. *Chem. Phys. Lett.*, 213:514, 1993. [43](#), [159](#)
- [109] E. van Lenthe, E.J. Baerends, and J.G. Snijders. *J. Chem. Phys.*, 101:9783, 1994. [80](#), [83](#)
- [110] L. Vočadlo and D. Alfè. *Phys. Rev. B*, 65:214105, 2002. [132](#), [133](#)
- [111] S.H. Vosko, L. Wilk, and M. Nusair. *Can. J. Phys.*, 58:1200, 1980. [41](#), [42](#)
- [112] F. Wagner, Th. Laloyaux, and M. Scheffler. *Phys. Rev. B*, 57:2102, 1998. [93](#)
- [113] R. Winkler. *Spin-Orbit Coupling Effects in Two-Dimensional Electron and Hole Systems*. Springer, 2003. [81](#)
- [114] Y. Zhang and W. Yang. *Phys. Rev. Lett.*, 80:890, 1998. [42](#)

## Following pages: Index

(this page inserted to enforce proper hyperlink to index)

# Index

- \$aims\_input, 256
- \$basis, 256
- \$coord, 256
- \$end, 259
- \$end, 260
- \$ener, 259
- \$estep, 259
- \$landauer, 256
- \$lsurc, 258
- \$lsurx, 258
- \$lsury, 258
- \$natoms, 256
- \$nlayers, 258
- \$nsaos, 257
- \$read\_omat, 257
- \$rsurc, 258
- \$rsurx, 258
- \$rsury, 258
- \$s1i, 258
- \$s2i, 258
- \$s3i, 258
- \$scfmo, 257
- \$self\_energy, 259
- \$testing, 260
- \$uhfmo\_alpha, 257
- \$uhfmo\_beta, 257
  
- abort\_opt, 28
- abort\_scf, 28
- Adams\_Moulton\_integrator, 201
- adaptive\_hartree\_radius\_th, 72
- aims\_control\_file\_template, 237
- aims\_restart\_string, 237
- anacon\_type, 161
- atom, 30, 134
- atom\_frac, 30
- auxil\_basis, 161
  
- basis\_threshold, 87
- batch\_distribution\_method, 201
- batch\_size\_limit, 58
- bfgs\_extrapolation\_cap, 113
- bfgs\_step, 113
  
- calc\_analytical\_stress\_symmetrized, 115
- calculate\_atom\_bsse, 48
- calculate\_perturbative\_soc, 82
- charge, 31
- charge\_mix\_param, 97
- check\_MD\_stop, 124
- clean\_forces, 113
- collect\_eigenvectors, 176
- communication\_type, 201
- compute\_analytical\_stress, 114
- compute\_forces, 114
- compute\_kinetic, 82
- compute\_numerical\_stress, 114
- constrain\_relaxation, 111
- constraint\_debug, 138
- constraint\_electrons, 138
- constraint\_it\_lim, 138
- constraint\_mix, 139
- constraint\_precision, 138
- constraint\_region, 137
- coulomb\_threshold, 172
  
- default\_initial\_moment, 97
- default\_max\_l\_prodbas, 162
- default\_max\_n\_prodbas, 162
- default\_prodbas\_acc, 161
- delta\_numerical\_stress, 114
- density\_update\_method, 68
- dft\_exe, 237
- distribute\_leftover\_charge, 72
- distributed\_spline\_storage, 176
- dos\_kgrid\_factors, 182
- dry\_run, 29
  
- elpa\_settings, 87
- empty, 46

- empty\_states, 88
- energy\_tolerance, 115
- evaluate\_work\_function, 182
- exx\_band\_structure\_version, 172
  
- fermi\_acc, 88
- final\_forces\_cleaned, 115
- fixed\_spin\_moment, 31
- force\_constants, 134
- force\_convergence\_criterion, 238
- force\_lebedev, 58
- force\_mpi\_virtual\_topo, 176
- force\_new\_functional, 202
- force\_occupation\_basis, 140
- force\_potential, 98
- force\_smooth\_cutoff, 202
- frequency\_points, 163
- frozen\_core, 35
- full\_embedding, 143
  
- grid\_partitioning\_method, 58
- grouping\_factor, 202
  
- harmonic\_length\_scale, 116
- hartree\_convergence\_parameter, 72
- hartree\_fourier\_part\_th, 73
- hartree\_fp\_function\_splines, 72
- hartree\_partition\_type, 73
- hartree\_radius\_threshold, 73
- hartree\_worksize, 202
- hessian\_block, 111
- hessian\_to\_restart\_geometry, 116
- hf\_version, 163
- homogeneous\_field, 142
- hse\_unit, 36
- hybrid\_xc\_coeff, 35
- hydro\_cut, 47
  
- ini\_linear\_mix\_param, 99
- ini\_linear\_mixing, 99
- ini\_linear\_mixing\_constraint, 139
- ini\_spin\_mix\_param, 99
- init\_hess, 116
- initial\_charge, 96
- initial\_ev\_solutions, 88
- initial\_moment, 96
- input\_template, 238
- k\_grid, 47
- k\_offset, 47
- k\_points\_external, 47
- KH\_post\_correction, 203
- KS\_method, 89
  
- l\_hartree\_far\_distance, 74
- lattice\_vector, 30, 134
- legacy\_monopole\_extrapolation, 74
- line\_search\_tol, 116
- line\_step\_reduce, 117
- load\_balancing, 176
- lopcg\_adaptive\_tolerance, 90
- lopcg\_auto\_blocksize, 90
- lopcg\_block\_size, 90
- lopcg\_preconditioner, 91
- lopcg\_start\_tolerance, 91
- lopcg\_tolerance, 91
  
- max\_atomic\_move, 117
- max\_lopcg\_iterations, 91
- max\_relaxation\_steps, 117
- max\_zeroin, 91
- maximum\_frequency, 164
- maximum\_time, 164
- mc\_int, 153
  - absolute\_accuracy, 155
  - kernel\_data, 155
  - number\_of\_MC, 155
  - output\_flag, 155
  - relative\_accuracy, 155
- MD\_clean\_rotations, 124
- MD\_gle\_A, 130
- MD\_gle\_C, 130
- MD\_maxsteps, 124
- MD\_MB\_init, 124
- MD\_restart, 125
- MD\_restart\_binary, 125
- MD\_run, 126
  - GLE\_thermostat, 129
  - NVE, 128
  - NVE\_4th\_order, 128
  - NVE\_damped, 128
  - NVT\_andersen, 128
  - NVT\_berendsen, 129
  - NVT\_nose-hoover, 130
  - NVT\_nose-poincare, 130
  - NVT\_parrinello, 129
- MD\_schedule, 126

- MD\_segment, 126
- MD\_thermostat\_units, 125
- MD\_time\_step, 127
- method, 239
- min\_batch\_size, 59
- min\_line\_step, 118
- mixer, 99
- mixer\_constraint, 139
- mixer\_swap\_boundary, 203
- mixer\_threshold, 101
- multip\_moments\_rad\_threshold, 74
- multip\_moments\_threshold, 74
- multip\_radius\_free\_threshold, 75
- multip\_radius\_threshold, 75
- multiplicity, 203
- multipole, 142
- multipole\_threshold, 75
  
- n\_anacon\_par, 165
- n\_atoms, 239
- n\_images, 239
- n\_iteration\_start, 239
- n\_max\_iteration, 240
- n\_max\_pulay, 101
- n\_max\_pulay\_constraint, 140
- n\_poles, 164
- n\_lcorr\_i\_leb, 157
- n\_lcorr\_nrad, 157
- numerical\_stress\_save\_scf, 118
  
- object\_function\_tolerance, 240
- occupation\_acc, 92
- occupation\_thr, 204
- occupation\_type, 92
- orthonormalize\_eigenvectors, 118
- output, 183
  - aitranss, 185
  - atom\_proj\_dos, 185
  - band, 186
  - band\_during\_scf, 186
  - basis, 187
  - cube, 188
  - density, 191
  - dipole, 191
  - dos, 192
  - eigenvectors, 193
  - elpa\_timings, 193
  - grids, 193
  - hessian, 194
  - hirshfeld, 194
  - k\_eigenvalue, 194
  - k\_point\_list, 195
  - matrices, 195
  - matrices\_parallel, 195
  - mulliken, 195
  - nuclear\_potential\_matrix, 196
  - ovlp\_spectrum, 196
  - quadrupole, 197
  - rho\_multipole, 197
  - species\_proj\_dos, 197
  - v\_eff, 198
  - v\_hartree, 198
  - zero\_multipoles, 199
- output\_in\_original\_unit\_cell, 183
- output\_level, 183
- override\_illconditioning, 93
- override\_relativity, 82
  
- packed\_matrix\_format, 177
- packed\_matrix\_threshold, 177
- partition\_acc, 59
- partition\_type, 60
- phonon, 232
  - band, 232
  - displacement, 232
  - dos, 232
  - free\_energy, 233
  - frequency\_unit, 234
  - supercell, 234
  - symmetry\_thresh, 234
- plus\_u\_petukhov\_mixing, 36
- points\_in\_batch, 61
- pole\_max, 165
- pole\_min, 165
- postprocess\_anyway, 101
- prec\_mix\_param, 102
- precondition\_max\_l, 103
- preconditioner, 102
- prodbas\_nb, 165
- prodbas\_threshold, 166
- prune\_basis\_once, 178
- pseudocore, 145
  
- qmmm, 143
- qpe\_calc, 37

- recompute\_batches\_in\_relaxation, 204
- relativistic, 83
- relax\_geometry, 119
- relax\_unit\_cell, 120
- restart, 103
- restart\_read\_only, 104
- restart\_relaxations, 121
- restart\_save\_iterations, 104
- restart\_write\_only, 104
- RI\_method, 162
  
- save\_data, 240
- sbtgrid\_lnk0, 166
- sbtgrid\_lnr0, 166
- sbtgrid\_lnrange, 166
- sbtgrid\_N, 166
- sc\_abandon\_etot, 105
- sc\_accuracy\_eev, 105
- sc\_accuracy\_etot, 105
- sc\_accuracy\_forces, 105
- sc\_accuracy\_potjump, 107
- sc\_accuracy\_rho, 106
- sc\_accuracy\_stress, 107
- sc\_iter\_limit, 107
- sc\_self\_energy, 37
- scgw\_it\_limit, 38
- scgw\_mix\_param, 38
- scgw\_print\_all\_spectrum, 38
- screening\_threshold, 173
- scs\_mp2\_parameters, 38
- set\_vacuum\_level, 76
- species, 31
  - nonlinear\_core, 146
  - angular, 62
  - angular\_acc, 62
  - angular\_grids, 62
  - angular\_min, 63
  - aux\_gaussian, 168
  - basis\_acc, 50
  - basis\_dep\_cutoff, 50
  - confined, 50
  - core, 50
  - core\_states, 51
  - cut\_atomic\_basis, 51
  - cut\_core, 206
  - cut\_free\_atom, 63
  - cut\_pot, 51
  - cutoff\_type, 52
  - division, 64
  - for\_aux, 168
  - gaussian, 53
  - hirshfeld\_param, 150
  - hydro, 53
  - include\_min\_basis, 54
  - innermost\_max, 64
  - ion\_occ, 54
  - ionic, 54
  - l\_hartree, 78
  - logarithmic, 64
  - mass, 33
  - max\_l\_prodbas, 169
  - max\_n\_prodbas, 169
  - nucleus, 33
  - outer\_grid, 65
  - plus\_u, 44
  - pp\_charge, 146
  - pp\_local\_component, 146
  - prodbas\_acc, 168
  - pseudo, 145
  - pure\_gauss, 55
  - radial\_base, 65
  - radial\_multiplier, 66
  - valence, 55
- spin, 31
- spin\_mix\_param, 108
- spring\_constant, 240
- spring\_constant\_max, 240
- spring\_constant\_min, 241
- squeeze\_memory, 204
- start\_BFGS, 241
- state\_lower\_limit, 167
- state\_upper\_limit, 167
- store\_EV\_to\_disk\_in\_relaxation, 178
- stress\_for\_relaxation, 120
- switch\_external\_pert, 108
- switch\_to\_CI-NEB, 241
- symmetry\_reduced\_k\_grid, 48
  
- thermodynamic\_integration, 133
- time\_points, 163
- total\_energy\_method, 39
  
- use\_2d\_corr, 40, 178
- use\_alltoall, 178
- use\_angular\_division, 204

---

use\_density\_matrix\_hf, [109](#)  
use\_dipole\_correction, [76](#)  
use\_hartree\_non\_periodic\_ewald, [77](#)  
use\_hf\_kspace, [173](#)  
use\_local\_index, [179](#)  
use\_logsbt, [167](#)  
use\_ovlp\_swap, [167](#)  
use\_pimd\_wrapper, [131](#)  
use\_variable\_spring\_constants, [241](#)

vdw\_correction, [149](#)  
vdw\_correction\_hirshfeld, [149](#)  
vdw\_method, [157](#)  
vdw\_pair\_ignore, [149](#)  
vdwdf, [153](#)  
    cell\_edge\_steps, [153](#)  
    cell\_edge\_units, [154](#)  
    cell\_origin, [153](#)  
    cell\_size, [154](#)

velocity, [123](#)  
verbatim\_writeout, [184](#)

walltime, [179](#)  
wave\_threshold, [48](#)  
wf\_extrapolation, [127](#)  
wf\_func, [127](#)

xc, [41](#)