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

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

A Users' Guide

FHI-aims team
with many contributors around the world.
April 28, 2023
https://fhi-aims.org/who-we-are

Contents

How to use this manual

If you are reading this introduction, you are likely reading the manual for the first time. In that case, please read on. There is, however, a strategy to use this manual most effectively to find keywords used in the input files to FHI-aims. This is it:

- Open the manual (pdf)
- Go to the table of contents
- At the bottom of the table of contents, click on "Index"
- Find the keyword you are looking for in the index
- Click on it.

Using the manual in this way may greatly reduce the barrier to looking up what a keyword actually does.

To first build FHI-aims, please also read this manual. You cannot simply type 'make'. Chapter 1, particularly sections 1.1–1.3, are what you need to read.

After this step, please also check out the online tutorials at

https://fhi-aims-club.gitlab.io/tutorials/tutorials-overview/

our web site

https://fhi-aims.org/

and the wiki, issue tracker and much more found at the FHI-aims gitlab site, to which everyone should have access:

https://aims-git.rz-berlin.mpg.de/

And now, for the actual ...

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) [110, 134, 60] 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 (e.g., GW and MP2), and wave-function based molecular total energy calculation based on Hartree-Fock and many-body perturbation theory (e.g., MP2, RPA, SOSEX, or the more encompassing renormalized second-order perturbation theory, RPT2).

Online tutorials for FHI-aims can be found at

https://fhi-aims-club.gitlab.io/tutorials/tutorials-overview/

or via a link at

https://fhi-aims.org/

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).

This is an Open Access paper and everyone should be able to obtain a copy at:
https://doi.org/10.1016/j.cpc.2009.06.022 .
Please cite this reference if you use FHI-aims.

However, FHI-aims is not just a product of this basic reference. Many more developments make this code a reality. For each individual FHI-aims run, a list of references describing the specific methods used is given at the end of the FHI-aims standard output. Please give credit in your publications if you can. FHI-aims is a scientific code, written by and for scientists. The primary recognition for their work is credit in the form of appropriate reference to their work.

Some particularly important papers (also worth reading!) follow below.

A list of methodological publications for specific methods in FHI-aims can also be found at

https://fhi-aims-club.gitlab.io/tutorials/basics-of-running-fhi-aims/

`references/`

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:

> A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, The Journal of Physics: Condensed Matter **26**, 213201 (2014).

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).

DFT calculations in FHI-aims employ density functionals from the Libxc library

> S. Lehtola, C. Steigemann, M. J. T. Oliveira, and M. A. L. Marques, Recent developments in LIBXC – a comprehensive library of functionals for density functional theory, SoftwareX **7**, 1 (2018).

which should be cited in any DFT paper.

Finally, we're quite proud that FHI-aims performed extremely well in the precision benchmark of 15 leading electronic structure codes known as the "Delta Project", https://molmod.ugent.be/deltacodesdft – see Reference [152] in Science Magazine for details. Numerical reliability – high precision – in everyday applications, applicable up to very large production problems – continues to be a top priority and is, in fact, one of the key reasons why FHI-aims was written in the first place.

In the present documentation, we do not repeat the 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, more on building the code from source, and we also address some issues ("troubleshooting") that are either beyond our control (operating-system related issues come to mind), or simply require some level of experience to address.

Electronic structure theory (and FHI-aims) is extremely versatile but many of the most interesting applications require complex workflows. We cannot possibly document them all on our own. Please consider sending us hands-on descriptions of any complex workflows that worked for you, and we would gladly include them in this manual (obviously, we'll happily include references to your work).

In any case, we hope that this manual 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.

Note: You cannot simply type 'make'. To find out what to do for a successful build, please look at sections 1.3-1.4, which will tell you what to do. There are a few performance related decisions that we cannot make for you on an unknown computer system, and the description below will hopefully help you make those decisions.

> Before you ask: FHI-aims is designed to run on any current Unix-based or Unix-like system, such as Linux or 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 subdirectories:

- *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 three levels of standard species defaults: "light", "tight", and "really_tight" (see Sec. 2.2). In addition, some further preconstructed special-purpose species defaults are provided in a "non-standard" subdirectory.

- *src/* : This directory, and its subdirectories, contain all of FHI-aims source code files.

- *testcases/* : Simple examples 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).

- *regression_tests/* : This directory contains a set of small standard test cases that can be run automatically using a script, `regressiontools.py` – when run without any flags, it will provide its own self-documentation. Unfortunately, running this script on a given platform and queueing system is not always trivial. If you can figure this out, we do recommend running and checking the regression tests on any new machine on which FHI-aims was installed. We have encountered rare but non-zero instances of compiler options (outside the control of FHI-aims) that produce correct numbers *almost* always – except for specific methods where the compiler has a bug. The regression tests will catch such issues before they strike in a production run. They will allow to check the compiled FHI-aims binary a little more extensively, but they are not strictly necessary to run FHI-aims. In particular, please do *not* view the input files of the regression tests as FHI-aims best practices. Follow the manual, not simply the regression tests. In many cases, they are not. Rather, what is tested may be a corner case that can be handled differently (better) in normal practical scenarios.

- *benchmarks/* : This directory contains specific example runs of calculations, including output files and specific timings, illustrating how FHI-aims should perform and scale on a current high-performance computer. They also include some essential practices to get high performance and memory efficiency in FHI-aims for large runs on very large computers. We highly recommend trying to run these benchmarks after successfully building FHI-aims on a parallel machine with sufficiently many CPUs. These benchmarks will give you an indication of whether you are achieving the expected performance of the code. This depends not only on building FHI-aims correctly, but also on the correct setup of the computing environment itself (not trivial). Running actual benchmarks is the best way to find out.

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

1.2 Prerequisites (libraries and software) you'll need

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 2003 (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/).

You should also have a version of GNU Make and CMake for compiling FHI-aims. If CMake is not present, it is also possible to work with just GNU Make, but it is worth the effort to obtain CMake. Typically, GNU Make will already be present on your system, either as `make`, or possibly as `gmake`. CMake should be available in the official repository of your Linux distribution.

The next two prerequisites are *optional*, but absolutely essential for any current use of FHI-aims: Support for parallel architectures, and (separately) support for fully parallel linear algebra. Thus, 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). Our experience is that Intel's MPI library is a very worthwhile investment on x86 platforms (better performance).
- 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).

Finally, the default compilation builds an executable which includes some parts of the code that are written in C. This may be turned off (see below), but we highly recommend compiling with C support as it introduces a number of useful features. You need:

- A C compiler – available on every Unix platform.

The creation of a complete, MPI-, scalapack-, and C-enabled binary is effort well spent. This should be the goal when compiling FHI-aims for any production purposes. This means that you should ultimately aim to build FHI-aims with the `USE_MPI` and `USE_SCALAPACK` CMake options enabled (see below).

To create an actually working FHI-aims build, please read sections 1.3, 1.4 and (for much more information) perhaps appendix H. Please also ask and consider helping out others by sharing settings that work on a given platform. This can be done via the FHI-aims slack channel or via the FHI-aims gitlab server – please see Section 1.7 below for ways to reach us.

1.3 Managing the build process with CMake

Building of FHI-aims is managed by CMake, which is a free and open-source build system generator. A build system generator is a tool that does not build anything by itself. Instead, it generates build scripts for a particular build system, e.g., Make, which are then used for the actual building. The build scripts, e.g., makefiles, are generated based on the user's environment and it is the job of CMake to ensure that the generation stage is as straightforward and failsafe as possible. In principle, CMake is completely platform agnostic (the C stands for cross-platform). The focus of the present is on supporting FHI-aims in a Linux or Unix environment.

CMake was released in 2000 and is currently used in a large number of projects (including some big ones like HDF5, KDE, mySQL, and Netflix). One of the motivators for FHI-aims was a push from the ESL (Electronic Structure Library) project to adopt CMake as the build management standard. ESL is a collection of electronic structure codes with the aim of avoiding duplication of functionality by connecting different electronic structure codes with each other with minimal effort. That is one of the reasons to use CMake as it makes it relatively easy to include other CMake projects into a given project.

1.3.1 Example CMake usage

Here is a typical example to get started with CMake.

1. Go to the root directory of FHI-aims (the top-level directory of the FHI-aims git repository or the distributed version of FHI-aims - i.e., one level above *src/*) and create a build directory:

   ```
   mkdir build && cd build
   ```

2. Create a file called `initial_cache.cmake` in the build directory or make a copy of `initial_cache.example.cmake` which is in the root directory. The following is example contents for that file,

   ```
   ################
   # Fortran Flags
   ################
   set(CMAKE_Fortran_COMPILER "mpif90" CACHE STRING "" FORCE)
   set(CMAKE_Fortran_FLAGS "-O3 -ip -fp-model precise" CACHE STRING ""FORCE)
   set(Fortran_MIN_FLAGS "-O0 -fp-model precise" CACHE STRING "" FORCE)
   ```

```
################
# C Flags
################
set(CMAKE_C_COMPILER "icc" CACHE STRING "" FORCE)
set(CMAKE_C_FLAGS "-O3 -ip -fp-model precise -std=gnu99" CACHE STRING "" FORCE)

################
# Libraries
################
set(LIB_PATHS "/opt/intel/mkl/lib/intel64" CACHE STRING "" FORCE)
set(LIBS "mkl_intel_lp64 mkl_sequential mkl_core
mkl_blacs_intelmpi_lp64 mkl_scalapack_lp64" CACHE STRING "" FORCE)

################
# Optional Flags
################

# Switch on/off use of mpi; default: ON
set(USE_MPI ON CACHE BOOL "" FORCE)
# Switch on/off use of scalapack; default: ON
set(USE_SCALAPACK ON CACHE BOOL "" FORCE)
set(USE_HDF5 OFF CACHE BOOL "" FORCE)
```

which you can edit to reflect your environment. The FORCE flag at the end of the set command tells CMake to overwrite existing entries. We recommend it as default. If you remove it, CMake will not change entries, once they have been initialized. When using the Intel C compiler, the `-std=gnu99` flag (`CMAKE_C_FLAGS` flags) is currently needed for the C sources of ELPA and i-PI (this would not be the case with gcc, which also works fine together with Intel Fortran).

As is evident, setting the correct values for these flags requires knowledge of several things: The Fortran and C compilers to be used, the Fortran and C compiler optimizations (or, correspondingly, flags for parts of the code that should not be optimized), the mathematical and MPI libraries to be used and their locations. Note that these are few items, but their choice is important for the performance of the code on a given computer. There are many different setups and automated tools do not always get these choices right. In the file above, we here identify those specific pieces where we feel that a user decision is necessary. Please ask for assistance (FHI-aims forums or slack channel) if needed.

3. Issue

   ```
   cmake -C initial_cache.cmake ..
   ```

   from the build directory to configure. In this example, the ".." directory is used. In general, the directory given in this command should point to the directory where the "CMakeLists.txt" file provided with FHI-aims is located.

   And yes – it has to be "`-C`" (capital C). "`-c`" (lowercase) will NOT work but will produce an error message that is not, unfortunately, helpful. So, if cmake refuses

to get to work at all, double-check the exact spelling of the above line first (and make sure that the "initial_cache.cmake" file is in place and that ".." indeed points to the correct directory).

If you encounter any other errors during this step, we recommend correcting your `initial_cache.cmake` file, saving it, then deleting the build directory and restarting from the first step.

4. Issue

   ```
   make -j [number]
   ```

   to build. An executable whose name starts with `aims` is created in the same directory.

   The value of `[number]` should be the same or less than the number of physical CPU cores available on your computer. Choosing sufficiently many cores speeds up the build process but on shared computers with multiple users (e.g., the login node of a cluster) it is typically nice to use only as many as you need, not necessarily the full node.

5. Move the newly generated FHI-aims binary to a directory where your binary files are typically collected. For example, if your FHI-aims top level directory contains a subdirectory *bin/*, use:

   ```
   mv aims.<version> ../bin
   ```

   In that command, replace the placeholder `<version>` with the actual completion of the name of the FHI-aims binary that you had just created.

For more details on how to use CMake, see Sec. H.

1.4 CMake variables

Here are some of the commonly used CMake variables.

- `CMAKE_Fortran_COMPILER` — Name of the Fortran compiler executable. Use a full path if location not automatically detected.
- `CMAKE_Fortran_FLAGS` — Compilation flags that control the optimization level and other features that the compiler will use.
- `LIB_PATHS` — List of directories to search in when linking against external libraries (e.g., "/opt/intel/mkl/lib/intel64")
- `Fortran_MIN_FLAGS` — Compilation flags only for files that should not be optimized because optimization is not needed. For example, the source file "read_control.f90" only controls how the input file `control.in` is read - but some compilers spend excessive amounts of time compiling this file if a different optimization level than "-O0" is specified.

- LIBS — List of libraries to link against (e.g., "mkl_blacs_intelmpi_lp64 mkl_scalapack_lp64")
- USE_MPI — Whether to use MPI parallelization when building FHI-aims. This should always be enabled except for rare debugging purposes. (Default: automatically determined by the compiler)
- USE_SCALAPACK — Whether to use Scalapack's parallel linear algebra subroutines and the basic linear algebra communications (BLACS) subroutines. It is recommended to always use this option. In particular, large production runs are not possible without it. The Scalapack libraries themselves should be set in LIB_PATHS and LIBS. (Default: automatically determined by LIBS)
- CMAKE_C_COMPILER — C compiler.
- CMAKE_C_FLAGS — C compiler flags.
- USE_LIBXC — Whether additional subroutines for exchange correlation functionals, provided in the LibXC library, should be used. By default, this is ON, i.e. LibXC will be compiled into the executable. It is advised to always use this. Please respect the open-source license of this tool and cite the authors if you use it.
- USE_SPGLIB — Whether the Spglib library for symmetry handling will be used. By deafult, this is ON, i.e. Spglib will be compiled into the executable. Please respect the open-source license of this tool and cite the authors if you use it.

For all CMake variables, see Sec. H.2.

For a detailed step-by-step cmake tutorial, please visit: https://aims-git.rz-berlin.mpg.de/aims/FHIaims/-/wikis/CMake%20Tutorial

1.4.1 MPI parallelization

On current computers, there is never a reason to compile FHI-aims without support for MPI in productions. Nevertheless, for testing purposes, it may sometimes be useful to compile without MPI support. We therefore cover this possibility here, also exemplifying how to manipulate CMake in a slightly more refined way.

In order to force MPI to be disabled, put

```
set(USE_MPI OFF CACHE BOOL "")
```

into the initial cache file. In order to force MPI to be enabled, use

```
set(USE_MPI ON CACHE BOOL "")
```

instead. If you want to enable/disable MPI support after the first configuration, issue

```
ccmake ~build
```

where ~build is the build directory. Move cursor to the field USE_MPI and hit enter. This toggles its state between ON/OFF. Hit 'c' to configure, 'g' to generate the build files, and rebuild the project.

1.5 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 *testcases/H2O-relaxation/* directory. The test run provided there relaxes a simple H_2O 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 *testcases/H2O-relaxation/* directory, type

```
../../bin/aims.version < /dev/null | tee H2O_test.own
```

at the command line. For "*version*", you must insert the code version stamp that was actually downloaded and built (for example, 171221 or whichever code version you are building).[1] For faster execution, 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. For 20 CPU cores, this could look like

```
mpirun -np 20 ../../src/aims.version < /dev/null | tee H2O_test.own
```

The result will be an output stream on your computer screen (created by "tee") which is also captured in an output file H2O_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.

> *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:

```
mpirun ../../bin/aims.version.scalapack.mpi.x < /dev/null > H2O_test.own
```

[1] The FHI-aims version stamp can be modified to whatever you wish in version_stamp.txt in the *src/* directory.

or even like this:

```
nohup mpirun ../../src/aims.version < /dev/null > H2O_test.own 2>&1 &
```

The latter version decouples the FHI-aims run completely from your current login shell 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 unadvisable 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 *testcases/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 and other numerical settings 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 ≈5 MB at a time when the typical available system memory 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. To avoid this, always 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" is prohibited by your computer (e.g., on MacOS), 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.

- Do not try to use OpenMP with FHI-aims unless you know exactly why you are doing this. FHI-aims is very efficiently MPI-parallelized and large portions of the code do not support OpenMP at all. (And they do not need to – MPI is simple as effective or more effective on practically all platforms in our experience.)

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!

1.6 Compiling faster versions of FHI-aims on specific platforms

FHI-aims is intended to be a Fortran-only code, which – for most of the code – means that building the "fastest" version of FHI-aims on a given computer architecture is "only" a matter of finding the right Fortran compiler and compiler options for that processor. For some architectures, specific compiler options are collected in the FHI-aims club wiki – please check there and please add any useful information that you may find.

That said, one particular performance-critical area for large systems is the Kohn-Sham eigenvalue solver. In FHI-aims and on parallel computers, this problem is solved by the ELSI infrastructure and the ELPA library. ELPA, in fact, allows its users to specify specific, platform-optimized so-called linear algebra "kernels." By default, FHI-aims uses a generic kernel which will compile with any Fortran compiler and will give reasonable speed. However, if one knows which specific computer chip one is using, it is possible to substitute this kernel with an architecture specific kernel and compile a faster version of ELPA into FHI-aims. This is possible, for example, for the BlueGene/P, BlueGene/Q, Intel AVX and several other Intel architectures. For standard Intel x86 chips, there is even an "assembler" based kernel that will get fast performance regardless of the Fortran compiler above.

Note that this choice can matter. For example, the "generic" ELPA kernel will produce fast code for the Intel Fortran compiler, but much slower code with certain versions of the PGI Fortran compiler (often found on Cray machines).

At this time, please ask (see below) about the most effective strategy to link against the "best" ELSI and ELPA libraries. Ideally, this will require a user to build a separate (standalone) instance of ELPA and of ELSI first. This can be very worthwhile.

1.7 Finding the other FHI-aims developers and users (talk to us!)

It can be surprisingly useful (and more fun) to find others who work with FHI-aims – to help find out who might already have solved a specific problem, how a given problem might be solved, exchange experiences, devise new and cool functionality that could take electronic structure theory to the next level, and so on. Some of us also simply like to have a cup of coffee with others (see below). FHI-aims only functions as a code and science tool because of the community around it, and we're always happy to meet new users, developers, and generally find out how to do better science together.

At the time of writing, we have a number of active communication channels. Anyone using or developing with FHI-aims is encouraged to frequent one or all of them:

- The forums and wiki at the FHI-aims gitlab server, https://aims-git.rz-berlin.mpg.de/ . This is a place where questions can be asked, answered, and looked up, and anyone is welcome and encouraged to share their experiences there. Ad-

ditionally, wiki entries are also encouraged. If nothing else, the wiki is a place to let the rest of the world know of successful build settings for FHI-aims on different platforms.

- An active *slack channel* (chat) at https://fhi-aims.slack.com/ . This is a place where a number of developers and users hang out and can be easily reached for questions in public semi-private and private conversations. Pretty effective. To join, you'll need an invitation from one of the slack channel owners, which we'll happily provide. Just ask, for by email (volker.blum@duke.edu is one of the owners, and there are several others as well).

- *Monthly FHI-aims video meetings* for anyone with an interest, usually announced on the FHI-aims slack channel.

- *FHI-aims Users' and Developers' meetings*, which we hold roughly every two years.

- For those who use the FHI-aims mainline (development) version – everyone with an FHI-aims license is welcome and encouraged to ask for access to this usually very stable version – there are nightly regression tests documented at https://aims-git.rz-berlin.mpg.de/ .

- Finally, for those who are shy, you are also welcome to *email* us:
  At aims-coordinators@fhi-berlin.mpg.de or (for those who are even more shy) email Volker, the lead developer, at volker.blum@duke.edu . Email is a productive avenue and Volker answers to the best of his abilities and available human time. However, bear in mind that one of the above channels will also reach Volker and, in addition, the many others who make FHI-aims happen and who might already have solved a problem and have an answer ... although you'd never have thought anyone did.

In short – please feel welcome and encouraged to talk to us if useful. FHI-aims is about science, and we're accessible. And if you're new to all this and someone helped you out especially, feel free to send them a Starbucks gift card (no one has ever done that, but hey, you could be the first :) or, even better, to cite their contribution to FHI-aims.

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 test case (relaxation of H_2O) described in Sec. 1.5.

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

As discussed in Sec. 1.5, 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.[1]

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. 1.5. 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>
```

[1] A few specific keywords (e.g., a restart of an existing calculation from an earlier wave function or density matrix) 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, 2017 : Test run input file control.in for simple H2O
#
##############################################################################
#
#  Physical model
#
  xc                 pbe
  spin               none
  relativistic       none
  charge             0.
#
#  Relaxation
#
  relax_geometry   bfgs 1.e-2
#
##############################################################################
#
#  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-processin
#  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 test case (relaxation of H_2O) described in Sec. 1.5. A section of *general* (system-wide) run-time 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.

For all elements, FHI-aims offers three or four different levels of *species_defaults*:

- *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*.

- *intermediate* : This level is presently only available for a few elements, but can play an important role for large and expensive calculations, especially for hybrid functionals. *Intermediate* settings use most of the numerical settings from *tight*, but includes basis functions between *light* and *tight*. The cost of hybrid functionals scales heavily with the number of basis functions found on a given atom. Full *tight* settings, which were designed with the much cheaper semilocal functionals in mind, can be prohibitively expensive for large structures and hybrid density functionals. Hybrid DFT results from *intermediate* settings are typically completely sufficient for production results, much cheaper, and we hope to produce *intermediate* defaults for a wider range of elements in coming years.

- *tight* : Regarding the integration grids, Hartree potential, and basis cutoff potentials, the settings specified here are rather safe, intended to provide meV-level accurate energy *differences* also for large structures. 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. For convergence purposes, the specification of the basis set itself (*tier* 1, *tier* 2, etc.) may still be decreased / increased as needed. Note that especially for hybrid functionals, *tight* can already be very expensive and specific reductions of the number of radial functions may still provide essentially converged results at a much more affordable cost (see *intermediate* settings).

- *really_tight* : Same basis sets (radial functions and cutoff radii) as in the *tight* settings, but for the other numerical aspects (grids, Hartree potential), settings that are *strongly overconverged* settings for most purposes. The idea is that *really_tight* can be used for very specific, manual convergence tests of the basis set and other settings – if really needed.
  Note that the *"tight"* settings are intended to provide reliably accurate results for most DFT production purposes; and they are not cheap. The absolute total energies for *tight* and DFT are in practice converged to some tens of meV/atom for most elements. To go beyond, take the *really_tight* settings and increase the basis set or other numerical aspects step by step. (Radial function by radial function may often be a good strategy to go.) *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.*
  Specific differences between *tight* and the unmodified *really_tight* settings: 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.
  Note that there can still be corner cases where you may want to test some numerical setting beyond *really_tight*. Mostly, these are custom scenarios or things beyond standard FHI-aims calculations of DFT total energies. Examples include: The confinement radius for surface work functions (should be checked), use of very extended or extremely tight Gaussian-type orbital basis functions (e.g., from very large Dunning-type basis sets – the density of the radial and angular grids should be checked), or RPA and MP2 calculations, which can need very different and often much larger basis sets (again, radial and angular grids should be checked).

The FHI-aims species defaults *light*, *tight*, *intermdiate*, *tight*, and *really_tight* are shipped in two versions in the folder *species_defaults*:

- *defaults_2010*
- *defaults_2020*

We recommend to use the re-worked *defaults_2020*. The updates of the *light*, *tight*, and *really_tight* defaults compared to the *defaults_2010* version are the results of a careful analysis of the Delta-Code DFT (DCDFT) Test (71 solids). So the updates should improve the accuracy of an element for the material class (insulator or metal) that is present in the DCDFT Test (e.g., the Be crystal is metallic in the test, but may be an ion for some other systems).

These are the current updates for the version *defaults_2020*:

- *light_spd*: The former light species defaults of *default_2010* for the elements 13-17, 31-35, and 49-53. These settings are of use for simulations, where the former defaults were sufficient and very light computational settings are needed (e.g. MD simulations).
- *light*: Be(+3p), Al-Cl(+4f), K(*radial_base*,*cut_pot*), Co-Ni(+ionic 4p), Ga-Br(+4f), Kr-Sr(*radial_base*,*cut_pot*), In-I(+4f), Xe-Ba, Rn(*radial_base*,*cut_pot*)
- *intermediate*: now, complete set of elements 1-86.
- *tight*: Mn(+3d function), Zn-Kr(+5g function), In-Xe(+5g function)
- *really_tight*: Zn-Kr(+5g function), In-Xe(+5g function)

A separate group of species defaults for light elements (H-Ar) is available especially for calculations involving explictly correlated methods (methods other than semilocal and hybrid density functionals):

- *NAO-VCC-nZ* : NAO type basis sets for H-Ar by Igor Ying Zhang and coworkers [244]. These basis sets are constructed according to Dunning's "correlation consistent" recipe. Their intended application is for methods that invoke the continuum of unoccupied orbitals explicitly, for instance MP2, RPA or GW. Note that they were constructed for valence-only correlation (hence "VCC", valence correlation consistent), i.e., they work best in frozen-core correlated approaches following a full s.c.f. cycle (core and valence) to generate the orbitals. While NAO-VCC-nZ can be used for "normal" density functional theory (LDA, GGA, or hybrid functionals), the normal "light", "intermediate", "tight" and "really_tight" species defaults are more effective in those cases. The advantage of NAO-VCC-nZ over GTO basis sets such as the Dunning "cc" basis sets is that with NAOs, both the behaviour near the nucleus as well as that for the tails of orbitals far away from atoms is much more physical. This means that we can use more efficient integration grids than for GTO basis sets to obtain systematic convergence of the unoccupied state space.

The *NAO-J-n* basis sets are designed for the calculation of indirect spin-spin coupling constants (J-couplings):

- *NAO-J-n* : The basis sets are available for most light elements from H to Cl. Since these are more expensive (tighter grids) than other basis sets, they should only be used for J-couplings. Even then, they should only be placed on atoms of interest, while cheaper basis sets can be used on other atoms. They are constructed by adding tight Gaussian orbitals to the NAO-VCC-nZ basis sets. In order to describe the Gaussian orbitals correctly near the nucleus, tighter grids than normally are required (with `radial_multiplier` 8 and `l_hartree` 8, among other parameters). Other stages of the calculation, such as geometry relaxation, should be performed with basis sets more suitable for the particular task (using, e.g., the default tight settings).

In addition, the *species_defaults* directory contains a few more sets of species defaults for special purposes. These can be found in the *non-standard* subdirectory and include:

- *gaussian_tight_770* : Species defaults that allow to perform calculation with some standard published Gaussian-type orbital (GTO) basis sets for elements H-Ar (including basis sets due to Pople, Dunning, Ahlrichs and their coworkers). These species defaults are meant to allow for exact benchmarks against GTO codes such as NWChem. The other numerical settings (especially grids) are thus much tighter than needed for "normal" NAO-type calculations. Note that FHI-aims is not optimized for GTO basis sets. We recommend NAO-type basis sets, not GTO basis sets, for production calculations – NAO-type basis sets are much easier to handle with our techniques and give better accuracy at lower cost. That said – the grid settings in the *gaussian_tight_770* species defaults are rather overconverged for benchmark purposes. One could create much more efficient species defaults for GTO basis sets – but GTOs still would not be as efficient as NAO basis sets (at the same level of accuracy).
- *Tier2_aug2* : Example, pioneered by Jan Kloppenburg, of basis sets that merge FHI-aims' tier2 basis sets with a very reduced set of Gaussian augmentation functions taken from Dunning's augmented correlation-consistent basis sets. This prescription appears to provide a remarkably accurate but affordable foundation to compute neutral (optical) vertical molecular excitation energies by linear-response time-dependent density functional theory, as well as (thanks to Chi (Garnett) Liu) the Bethe-Salpeter Equation.
- *light_194* : This is just an example of how to tune down the normal "light" basis sets of FHI-aims by reducing the integration grid even further. For things like fast molecular-dynamics type screening of many structures, this is a perfectly viable approach. Examples are provided for H-Ne. Obviously, do test the impact of such modifications for your own purposes.

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

To extrapolate the absolute total energy of methods which rely on the unoccupied state continuum explicitly, e.g., RPA or MP2, we recommend using the NAO-VCC-nZ basis sets. These basis sets are presently available for light elements (H-Ar). A popular completeness-basis-set extrapolation scheme is two-point extrapolation:

$$E[\infty] = \frac{E[n_1]n_1^3 - E[n_2]n_2^3}{n_1^3 - n_2^3}$$

where "n_1" and "n_2" are the indicies of NAO-VCC-nZ. This $1/n^3$ formula was originally proposed for the correlation energy, but was also used directly for the total energy.

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*, *intermediate*, *tight*, and (if ever needed!) *really_tight*, should provide such reliable accuracy as they come. The *NAO-VCC-nZ* basis sets provide additional functionality specifically for correlated methods (MP2, RPA, GW, etc.) and light elements. However, in all *species_default* files, all important accuracy choices are 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, but may be very expensive for hybrid functionals. Look to the *intermediate* settings for a more economical choice in that case.

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, Ref. [28]) 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 electon 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 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_base` 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.

For an illustration of the effect of the `radial_multiplier` keyword on the density of the radial grid shells, go to Ref. [244] (http://iopscience.iop.org/1367-2630/

15/12/123033/article, open access) and look at Figure A.1 and the accompanying explanation.

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 Why does my calculation take too long?

This is, indeed, an excellent question to ask. Understanding what the code spends its time on, and why, is often the best first approach to understanding what is actually being calculated – and thus, to learn something about the scientific problem to be solved.

Many calculations take as long as they do, simply because getting an accurate result for many atoms can take some time.

That said, if calculations that seemed simple start taking excessive amounts of time, it may be a very good idea to question your input settings. It may also be a very good idea to actually read the output of the code. It tells you a lot about what the code does. Some suggestions for different scenarios:

- Do invest the time to compile a scalapack enabled binary and actually use scalapack. Most architectures today are parallel, and using those efficiently is perhaps the single biggest technical strength of FHI-aims. Never ask for the use of a lapack eigenvalue solver (the serial version, i.e., the eigenvalue solver that only uses a single CPU) explicitly unless you are testing. The code sets the correct default automatically if needed. But if you ask for the serial eigenvalue solver explicitly, you may find 999 of your 1000 CPU cores doing nothing. (See the keyword `KS_method` . Most importantly, never set this keyword explicitly if there is no reason to do so.)
- Look at the timing output at the end of each s.c.f. iteration – not(!) just the final timings. These timings summarize the time spent for each of the physical steps of your calculation, and can tell you a great deal about what is going on. Here's an example where something went wrong:

```
End self-consistency iteration #     1       :  max(cpu_time)    wall_clock(cpu1)
| Time for this iteration                     :      219.302 s         219.900 s
| Charge density update                       :       16.121 s          16.162 s
| Density mixing & preconditioning            :        1.084 s           1.099 s
| Hartree multipole update                    :        0.088 s           0.090 s
| Hartree multipole summation                 :        4.440 s           4.489 s
| Integration                                 :        0.980 s           0.986 s
| Solution of K.-S. eqns.                     :      196.568 s         197.023 s
| Total energy evaluation                     :        0.004 s           0.016 s
```

The key times to look for here are the wall clock times. This is the physical time spent by the code on each task. The individual sub-timings of each task should roughly add up to the total time, which is given first.

The "CPU time", on the other hand, is measured internally, without accounting for times when the CPU is in fact idle. The CPU time is only given here since large deviations between wall clock time and internally measured CPU time are a good way to indicate an inefficient computer system setup. In case of doubt, however, wall clock time is the relevant measure for the real cost of the calculation

Typically, the time for the density update should be approximately the same (within a single-digit factor) as for the creation of the Hamiltonian integrals. (All these numerical steps are explained in the FHI-aims CPC paper, Ref. [28].) The fact that this is not the case indicates some kind of a problem.

However, the bulk of the time is spent in what is called "Solution of K.-S. eqns.", which here means the simple solution of a matrix eigenvalue problem. This is simple linear algebra. This step scales formally as $O(N^3)$ with system size N, while all other steps scale roughly as $O(N)$.

This means that the eigenvalue problem should become the dominant part of the calculation time only for rather large systems (100s or 1,000s of atoms, depending on whether heavy or light elements are used, whether the system is non-periodic or periodic, etc.). The fact that the eigenvalue problem takes up so much time above warrants at least a question.

In the case shown above, a periodic calculation was conducted, with a total of 64 k-points, i.e., a total of 64 independent eigenvalue problems to be solved. Asking for many k-points is obviously a good reason why the solution of eigenvalue problems could dominate.

In the case considered here, however, the number of basis functions (the matrix dimension) was only a few thousand.[2] As a rule of thumb, this should not have been a problematic matrix size yet. (Several ten thousand basis functions or, perhaps, a few thousand k-points are typically what is needed to make a single eigenvalue solution become relevant, even on a large number of CPU cores.)

What happened above is that the calculation was in fact conducted in parallel on ≈500 CPU cores, but erroneously enforced a serial eigenvalue solver in the `control.in` file. This means that about 450 CPU cores idled while the eigenvalue problem was solved on only a few others.

The point of this example is: It helps to check and question the timing output. Another common problem is the fact that the calculation of forces costs far more than just the calculation of the electron density. Thus, the FHI-aims code by default only computes forces once the s.c.f. cycle is otherwise converged. If, however, your s.c.f. convergence criteria are set inadequately, you might see ten s.c.f. iterations per geometry step computing forces. The code has no way to foresee this, but as a user, you may be able to check after the fact, and prevent this behavior for the future.

- Mixing factor and occupation broadening. These are again values that decidedly depend on the system type to be computed, which is difficult to foresee from the perspective of the code. The default values for `charge_mix_param` and `occupation_type` are set somewhat automatically by the `adjust_scf` keyword, according to whether or not the system is estimated to have a HOMO-

[2] It is truly a system-dependent question what constitutes "many" k-points. For example, a metallic system with a single atom per unit cell should not have much trouble with, say, 24^3 k-points. On the other hand, a 1,000 atom supercell should probably not use more than a single-digit number of k-points. In the specific case considered above, 64 k-points did not happen to be a particularly large number.

LUMO gap. However, tweaking these values is possible. For instance, for metals, `occupation_type` 0.1 eV is often very reasonable.

- There are (obviously) the numerical convergence parameters of the preceding section that should be heeded. For example, "tight" settings can be much more costly than "light" settings. Obviously, "tight" settings are needed for accurately converged final numerical results in many cases. However, this does not mean that, e.g., a long pre-relaxation has to be done with "tight" settings – prerelaxing with "light" settings and switching to "tight" settings only then is usually the way to go. Also, consider the "intermediate" settings where available, especially for hybrid density functionals.

- Exchange-correlation methods beyond LDA and GGA typically take much more time. Here, the key bottleneck is the evaluation of the two-electron Coulomb Operator and its manipulations later. Even then, it pays to spend time learning about the respective settings, for instance, the `RI_method` to be used, the internal thresholds that go with it, or whether it is possible to reduce the number of s.c.f. steps in some other way.

2.5 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.

2.6 Unit conventions and fundamental constants in FHI-aims

FHI-aims' output files provide lengths, energies and other output quantities. For comparison purposes, especially to other codes, it is very important to understand the origin of the units used in FHI-aims and what they mean. There are two key distinctions:

- **Internally**, i.e., in the actual math performed in the code (unlike in the output), FHI-aims uses "atomic units," specifically, Hartree atomic units. At the time of writing of this manual, a more detailed explanation of Hartree atomic units can be found on Wikipedia:
  `https://en.wikipedia.org/wiki/Hartree_atomic_units`.
  In detail, Hartree atomic units mean that Planck's constant, the elementary charge, the Bohr radius, and the electron mass all take values of unity (i.e., 1). In this notation, Schrödinger's equation or Dirac's equation simplify to mathematical expressions that are universally comparable between different electronic structure codes. When provided with the same problem in atomic units, all of them should give the exact same answer in atomic units.
- **Externally**, that is in the input and output files of FHI-aims, the expected unit convention is to always provide lengths in Å, energies in eV, and various other derived units instead of the underlying atomic units.
  These derived values depend on the conversion factors used.
  While these conversion factors are standardized, they are in fact the result of experimental measurements. Standard tabulations are provided by standards bodies such as the National Institute of Standards (NIST), but these tabulations are updated every few years. Thus, the "value" of one electron volt (or other derived units) may vary slightly between different electronic structure codes, whereas the internal atomic units are always the same.

In FHI-aims' output, some quantities (e.g., energies) are therefore frequently provided both in terms of atomic units and in terms of more conventional units from other unit systems. For instance, energies are provided in Hartrees (unassailable units from the Hartree atomic unit system) as well as in eV (derived unit, using a conversion factor).

As of this writing (2022), FHI-aims uses the NIST CODATA 2002 (Web Version 4.0 2003-12-09) unit conventions for key conversion factors, most importantly

- 1 Ha = 27.2113845 eV,
- 1 Bohr radius = 0.52917721 Å.

The exact conventions can be found in the file `src/constants.f90` in the FHI-aims source code. And yes, the source code can be read – it is human readable – and we encourage any user of FHI-aims to do so in case of uncertainty regarding the units. This, the source code, is where mathematical decisions are ultimately made and the unit conventions are very simple to check. Additionally, the value of the "Hartree" unit used (in eV) is also printed near the end of each FHI-aims standard output.

We note that other codes can and do use other unit conventions. For example, the frequently used Atomic Simulation Environment (ASE) uses CODATA 2014 by default at the time of writing (ASE versions 3.22/3.23). This version difference entails a small but noticeable change in the definition of the electron volt value, after conversion from Hartree units. Total energy differences as well as small energy eigenvalues will not be significantly affected by these changes, as long as consistent units are used everywhere. However, mixing different unit conventions in differences of large numbers can lead to large errors and should be avoided – always stick to a single unit convention.

For the time being, FHI-aims continues to follow the unit convention selected at its inception, since this choice guarantees continuity of results between different versions of FHI-aims. However, please be aware of the unit conventions described in the present section.

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. 2.5 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., afer 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. 2.5 for some further remarks.

Tag: `control.update.in` (file)

Usage: Allowed content of this file is a fairly limited subset of what is allowed and parsed in control.in . Details below.

Purpose: FHI-aims checks for presence of this file in the current working directory at the end of each individual iteration of the SCF cycle. If the file is found, it is parsed, and found settings are updated. Note that if you do not remove the file manually, it will be parsed after each iteration. But with the current limited functionality, this should not pose any problems.

This file allows to modify some of the parameters of a calculation during runtime of FHI-aims. At present, this is limited to the settings of the convergence of the SCF cycle, namely: sc_accuracy_rho , sc_accuracy_eev , sc_accuracy_etot , sc_accuracy_potjump , sc_accuracy_forces , sc_accuracy_stress .

Tags for general section of `control.in`:

Tag: `check_cpu_consistency` (control.in)

Usage: `check_cpu_consistency flag`

Purpose: In parallel runs, determines whether the consistency of geometry-related arrays is verified explicitly between different MPI tasks.

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

This flag is introduced as default purely to monitor and possibly undo errors that should not happen. Theoretically, all MPI tasks of a given FHI-aims run should have the same atomic coordinates and lattice vectors. In practice, it appears that certain hardware and/or compilers/libraries introduce bit flips between different instances of what is formally the same variable on different CPUs.

If `check_cpu_consistency` is `.true.`, the code checks for deviations.

If the discrepancy is numerically negligible (below the value set by the tolerance parameter `cpu_consistency_threshold`, the code will work based on the assumption that the observed discrepancy is a platform-dependent artifact, will set all instances of the geometry to that stored on MPI task myid=0, and continue the run. Nonetheless, a warning will be printed in the output file and near the end of the output.

If the discrepancy is larger than the tolerance parameter `cpu_consistency_threshold`, the code will stop and inform the user.

Tag: `cpu_consistency_threshold` (control.in)

Usage: `cpu_consistency_threshold tolerance`

Purpose: In parallel runs, determines the degree to which inconsistencies of geometry-related arrays will be tolerated between different MPI tasks.

`tolerance` : A small real numerical value, positive or zero. Default: 10^{-11}.

See keyword `check_cpu_consistency`. If `check_cpu_consistency` is `.true.`, then keyword `cpu_consistency_threshold` determines the maximum value to which discrepancies of geometry-related quantities between different MPI tasks will be tolerated (they will, however, be set to identical values even if the run continues).

Tag: `check_stacksize` (control.in)

Usage: `check_stacksize flag`

Purpose: Determines whether a check of stack size is performed.

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

By default, FHI-aims checks that unlimited allocation on stack are allowed by the operating system. This option allows to disable this check.

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.

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

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

Usage: `cite_reference string`

Purpose: Triggers the addition of a specific citation to the end of the FHI-aims standard output for a given run.

`string` is a string that identifies the reference in question.

This feature is useful, e.g., to make sure that the literature reference for a given basis set (encoded in the species_defaults input file) is available at the end of an FHI-aims run.

Each citation must, however, be coded into the FHI-aims source code in module `applicable_citation` to ensure that the requested output is actually available. Note that the practical format for such references can vary widely – from a simple string (explanation who did the work) all the way to the more usual case of a journal reference.

At the time of writing, species-related legitimate values of `string` are:

- `NAO-VCC-2013` for reference [244], describing the NAO-VCC-nZ basis sets for valence-correlated calculations of elements H-Ar (useful for basis set extrapolation for many-body perturbation methods, e.g., MP2, RPA, RPT2, or GW).

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-valued numbers (in Å) which specify the atomic position. `species_name` is a string descriptor which specifies the chemical element (or, more broadly, atomic species type) at this atomic position; it must match one of the `species` descriptions defined in `control.in`.

Tag: `atom_frac` (geometry.in)

Usage: `atom_frac` f_1 f_2 f_3 `species_name`

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

f_i is a real-valued multiplier to `lattice_vector` i. `species_name` is a string descriptor which specifies the chemical element (or, more broadly, the atomic species type) at this atomic position; it must match one of the `species` descriptions defined in `control.in`.

Fractional coordinates are only meaningful in periodic calculations.

Conversion of fractional atomic positions into cartesian coordinates, $\boldsymbol{R}_I$, is achieved by

$$\boldsymbol{R}_I = \sum_{i=1}^{3} f_i \cdot \boldsymbol{a}_i, \tag{3.1}$$

where $\boldsymbol{a}_i$ is a unit cell vector specified by the `lattice_vector` keyword.

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.

Warning: It is not a good idea to run each and every calculation with `spin collinear` just because that seems to be the more general case. In a system that will safely be non-magnetic, using something other than `spin none` *will* roughly double the CPU time needed in the best case, and it will most likely lead to much worse s.c.f. convergence (i.e., more iterations needed to find the self-consistent electronic solution). There is no fundamental problem with running `spin collinear`, but again: just doing this out of some sense of impartiality may not be a wise use of resources.

This keyword is completely independent of spin-orbit coupling, which is applied as a post-processed correction after the SCF cycle has converged. The two keywords can be used together, e.g., to obtain spin-orbit coupled versions of the energy band structure of an open-shell system (see the supporting material of Ref. [113] for an example of the spin-polarized band structure of fcc Ni). For more information on spin-orbit coupling, please see the `include_spin_orbit` keyword and the discussion in the associated chapter.

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 a real number (the nuclear charge).

`Z` is usually an integer number. However, partial (non-integer) charges are also possible.

Fractional `Z` can be useful, for example, for a stoichiometric hydrogen-like termination of a compound semiconductor slab (e.g., in a III-V compound, the valence of the connecting element would be mimicked by a fractionally charged H of charge 0.75 or 1.25).

If the difference between the specified nuclear charge and the nearest integer is greater than 0.34, keyword `element` may be needed to be set explicitly in the species definition to designate an unambiguous chemical identity.

Fractional `Z` can also be useful to distribute a compensating charge for an electronically charged periodic supercell calculation. In electronic charged periodic systems, a compensating background charge is always implicit. This is often accomplished by introducing an implicit homegeneous charged background density. However, the choice of such a jellium background is often anything but ideal. For instance, in a surface slab calculation, part of this compensating charge will be located in the vacuum. In such cases, it may be better to place the compensating charge in the system explicitly and "by hand". One good way to do this is to place the compensating charges on certain nuclei.[200]

If you add a fractional `Z` to a species_default, you will have to take care to modify the `valence` tags to reflect the exact opposite charge, creating an overall neutral spherical free atom as far as the `valence` occupation numbers in the species definition go.

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

Usage: `element symb`

Purpose: Chemical element associated with the species.

`symb` is a string (max. 2 characters) that corresponds to the symbol of the chemical element. Default: see below.

The purpose of this tag is to specify the chemical identity of a species in the rare cases when it cannot be determined from `Z` because it has been set to a non-integer value. In particular, when `Z` is more than 0.34 from the nearest integer number, the species element must be set with the `element` tag. Currently, this value is used only by the van der Waals routines, but the requirement above must be satisfied for any calculation.

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. 3.23 and Ref. [196].

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. [196]. 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:

$$E_{\text{tot}}^{\text{RPA}} = E_{\text{tot}}^{\text{DFT}} - E_{\text{xc}}^{\text{DFT}} + E_{\text{x}}^{\text{EX}} + E_{\text{c}}^{\text{RPA}} \,. \tag{3.2}$$

$E_{\text{tot}}^{\text{DFT}}$ 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_{x}^{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 [198] and the second-order screened exchange (SOSEX) correction [87]. The renormalized SE (rSE) and SOSEX corrections can be combined. The combined scheme is called "renormalized 2nd-order perturbation theory" (rPT2) [197],

$$E_{\text{tot}}^{\text{rPT2}} = E_{\text{tot}}^{\text{RPA}} + E_{\text{c}}^{\text{SOSEX}} + E_{\text{c}}^{\text{rSE}} \,. \tag{3.3}$$

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 [7].

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).

- 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: `frozen_core_postscf` (control.in)

Usage: `frozen_core_postscf valence_shell_number`

Purpose: Alternative way to specify the valence shells in the frozen-core algorithm for MP2, RPA and rPT2 methods. *This keyword is valid for elements from H (1) to Rn (86).*

`valence_shell_number` is the number of valence shells which are taken into account in the frozen-core MP2, RPA or rPT2 calculations.

Compared to the keyword `frozen_core`, `frozen_core_postscf` is more friendly, especially for large systems, as you don't need to count by hand which is the first valence orbital in the frozen-core algorithm.

For example, *valence_shell_number=2* means that at most two outer shells are taken as valence shells in the frozen-core calculations:

element	core shells	valence shells
H, He	–	$1s$
Li-Ne	–	$1s2s2p$
Na-Ar	$1s$	$2s2p3s3p$
K-Kr	$1s2s2p$	$3s3p3d4s4p$

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, α=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 Å^{-1}) or 'b' or 'B' (for [bohr radius]$^{-1}$).

The `hse06` functional comes with a screening parameter ω 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 ω – either $Å^{-1}$ or [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: `lc_dielectric_constant` (control.in)

Usage: `lc_dielectric_constant value`

Purpose: If set, will modify the amount of exact exchange in the hybrid XC functional LC-ωPBEh.
`value` is a real number (larger or equal to one) that specifies the degree of exact exchange admixture in the long-range part. default=1.0

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 [189].

`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. [189].

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, assums a maximal effect of the DFT+U term on the occupation numbers, handling double counting correctly in the case that all orbitals with in the shell are either fully occupied or empty. The self consistent mixing of both limits improves the handling of the intermediate range (see Ref. [189]).

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. The eigenvalues are iterated in G **and** W. Molecular orbitals are kept unchanged from the preliminary calculation. This scheme is often abbreviated as evGW in literature. For true self-consistent GW, see the `sc_self_energy` further below.
- `ev_scgw0` Perturbative G_0W_0-type self-energy, where self-energy is evaluated with partial self-consistency in the eigenvalues. The eigenvalues are iterated **only** in G, but not in W. Molecular orbitals are kept unchanged from the preliminary calculation as in ev_scgw. This scheme is often abbreviated as evGW_0 in literature. 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.

For more details on different GW flavors (G_0W_0, partially and fully-selfconsistent schemes) see review article [80].

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` α

Purpose: Define the linear mixing coefficient α, 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: `rpa_along_ac_path` (control.in)

Usage: `rpa_along_ac_path rpa_along_ac_path_grid`

Purpose: Calculate the RPA-approximated potentials along the adiabatic-connection path.

`rpa_along_ac_path_grid` is the number of potentials you want to sampling along the adiabatic-connection path.

The standard RPA method is a adiabatic-connection advanced DFT method, which integrates the contribution along the adiabatic-connection path analytically. This keyword `rpa_along_ac_path` allows you to unpack the adiabatic-connection path in the RPA approximation in detail.

Tag: `printout_dft_components` (control.in)

Usage: `printout_dft_component given_dft_method`

Purpose: Evaluate the XC contributions of a given DFT method based on SCF converged KS (or HF) orbitals.

`given_dft_method` is the name of the DFT method you want to investigate. At present, only two GGA methods (PBE and BLYP) are avaiable.

This keyword `printout_dft_components` is repeatable in the contril.in allowing to inspect several DFT methods in one task.

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:

$$E_{\text{corr,MP2}} = E_T + E_S \,. \tag{3.4}$$

Grimme [86] 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:

$$E_{\text{SCS,MP2}} = p_T E_T + p_S E_S \,. \tag{3.5}$$

For example, p_T=1/3 and p_S=6/5 are employed to obtain the reaction energies of Table I in Ref. [86].

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).

All LDA, GGA and meta-GGA DFT functionals listed under `xc` can be used with `total_energy_method` , including those implemented through `LibXC` and `dfauto`; *See*

the relevant section of `xc` *options for more detail.* Hybrid-DFT functionals are also available, with the exception of range-separated or long-range corrected approaches.

Other 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.
- `ll_vdwdf` : The nonlocal part of correlation energy is calculated using the van der Waals density functional proposed by M. Dion *et al.* [59] 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. 3.22. Note that an alternative implementation by the Helsinki group is available as well, the present keyword is not your only option.
- `mp2` : The correlation energy is calculated in second-order Møller-Plesset perturbation theory (MP2), with Hartree-Fock added for the exchange part. *Note added in March 2016: A periodic implementation of MP2 is available but, at the time of writing, computationally extremely expensive. If you decide to use it, please keep in mind that the periodic version is included here as a matter of protocol but is not yet optimized to be fully usable in production calculations.*
- `pbe_vdw` : Evaluates the van der Waals density functional proposed by M. Dion *et al.* [59] with the methodology of Sec. 3.22.2. (Uses PBE exchange.) *This is* not *the Tkatchenko-Scheffler correction [224]. If you are looking for Tkatchenko-Scheffler, please use the keyword* `vdw_correction_hirshfeld` *instead.*
- `revpbe_vdw` : As pbe_vdw but uses revpbe instead of pbe for the exchange. *This is also* not *the Tkatchenko-Scheffler correction [224]. If you are looking for Tkatchenko-Scheffler, please use the keyword* `vdw_correction_hirshfeld` *instead.*
- `nlcorr` : Only the non-local correlation term of the pbe_vdw or revpbe_vdw is calculated and added to the total energy. *And this is still* not *the Tkatchenko-Scheffler correction [224]. If you are looking for Tkatchenko-Scheffler, please use the keyword* `vdw_correction_hirshfeld` *instead.*
- `rpa` : The RPA total energy as defined in Eq. (3.2) will be calculated. When this option is specified, the SE and rSE corrections to RPA are also evaluated. The total enegies computed with the RPA, RPA+SE, and RPA+rSE schemes are listed in items `"RPA total energy"`, `"PRA+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. (3.3) 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.
- xyg3 : "XYG3" double-hybrid functional[?], which is defined only for a self-consistent B3LYP reference, i.e., xc b3lyp is mandatory. *Note* that double-hybrid functionals include MP2 components. When using the *tier* basis sets, you must use a counterpoise correction of energy differences to get reliable results.
- xdh-pbe0 : "xDH-PBE0" double-hybrid functional[245], which is defined only for a self-consistent PBE0 reference, i.e., xc pbe0 is mandatory. *Note* that double-hybrid functionals include MP2 components. When using the *tier* basis sets, you must use a counterpoise correction of energy differences to get reliable results.

Note that some of the correlation methods available here are only supported for cluster geometries at this time. *Note* also that when advanced correlation methods (e.g. rpa, rpt2, xyg3, xdh-pbe0 and mp2) are used for binding energy calculations, a **counterpoise correction** should always be performed with the default NAO basis sets in FHI-aims to get reliable results, since the basis set superposition error (BSSE) for these correlation methods is significant. For these advanced correlation methods, the sequence of NAO valence-correlation consistent basis sets (*NAO-VCC-nZ*[244]) is a better choice, which reduces the basis set incompleteness error, including BSSE, with increasing the basis size, and especially enables to approach the completeness-basis-set limit with the aid of extrapolation scheme.

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. See also xc_pre .

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 approximation (different parameterizations):
  - `pw-lda` : Homogeneous electron gas based on Ceperley and Alder [42] as parameterized by Perdew and Wang 1992 [183]. *Recommended LDA parameterization.*
  - `pz-lda` : Homogeneous electron gas based on Ceperley and Alder [42], as parameterized by Perdew and Zunger 1981 [184].
  - `vwn` : LDA of Vosko, Wilk, and Nusair 1980 [233].
  - `vwn-gauss` : LDA of Vosko, Wilk, and Nusair 1980, *but based on the random phase approximation* [210]. 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 [9]
  - `blyp` : The BLYP functional: Becke (1988) exchange [20] and Lee-Yang-Parr correlation [150].
  - `pbe` : GGA of Perdew, Burke and Ernzerhof 1997 [181].
  - `pbeint` : PBEint functional of Ref. [67]
  - `pbesol` : Modified PBE GGA according to Ref. [187].
  - `rpbe` : The RPBE modified PBE functional according to Ref. [93].
  - `revpbe` : The revPBE modified PBE GGA suggested in Ref. [247].
  - `r48pbe` : The mixed functional containing 0.52*pbe and 0.48*rpbe according to Ref. [173]
  - `pw91_gga` : GGA according to Perdew and Wang, usually referred to as "Perdew-Wang 1991 GGA". This GGA is most accessibly described in References 26 and 27 of Ref. [182]. Note that the often mis-quoted reference [183] does *not*(!) describe the Perdew-Wang GGA but instead only the correlation part of the local-density approximation described above.
  - `b86bbpe` : Becke's B86b exchange[19] plus PBE correlation[181].
- Meta-generalized gradient approximations:
  - `m06-l` : Truhlar's optimized meta-GGA of the "M06" suite of functionals. [250]
  - `m11-l` : Truhlar's optimized range-separated local meta-GGA of the "M11" suite of functionals. [191]

  - `revtpss` : Meta-GGA revTPSS functional of Ref. [185, 186].
  - `tpss` : Meta-GGA TPSS functional of Ref. [223]
  - `tpssloc` : Meta-GGA TPSSloc functional, thanks to E. Fabiano and F. Della Sala. L.A. Constantin, E. Fabiano, F.Della Sala, Ref. [51].
  - `scan` or `SCAN`: "Strongly Constrained and Appropriately Normed Semilocal Density Functional," i.e., the SCAN meta-GGA functional by Sun, Ruzsinszky, and Perdew.[219]

- Hartree-Fock and hybrid functionals (including non-local exchange): *Please also see Secs. 3.23 and 3.24 for related keywords and technical hints.*
  - `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. [233, 210] for details). Note that this is therefore *not* exactly the same B3LYP as originally described by Becke in 1993.
  - `hf` : Hartree-Fock exchange only.
  - `hse03` : Hybrid functional as used in Heyd, Scuseria and Ernzerhof [105, 106]. 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. [141, 106], 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: $\omega_{\mathrm{HF}} = 0.15/\sqrt{2}$
    Screened PBE-like exchange: $\omega_{\mathrm{PBE}} = 0.15 \times 2^{1/3}$
    Following the notation of Ref. [141], the 'hse03' functional in FHI-aims reproduces these original values exactly.
  - `hse06` : Hybrid functional according to Heyd, Scuseria and Ernzerhof [105], following the naming convention suggested in Ref. [141]. In this case, the additional option `value` is needed, representing the single real, positive screening parameter `omega` (ω) as clarified in Ref. [141]. 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 ω is either $Å^{-1}$ or (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 $Å^{-1}$ ('A') or 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-1`[1]
    or similar.

[1] The `hse_unit` flag reads only the first character. Thus this is equivalent to `hse_unit b` (case insensitive).

A few comments on typical choices for ω in the earlier literature:
The original value of 0.15 bohr^{-1} by Heyd, Scuseria and Ernzerhoff 2003 [105] was never true - see their 2006 erratum. In FHI-aims, the 'hse03' functional implements their actual choice.
Krukau, Vydrov, Izmaylov and Scuseria 2006 [141] clarify the distinction between 'hse03' and 'hse06' (in addition to the Erratum mentioned above). Their conclusion is that omega=0.11 bohr^{-1} is a reasonable choice.
Vydrov, Heyd, Krukau and Scuseria in 2006 [179] appear to favor omega=0.25 bohr^{-1}, but with a mixing parameter (keyword `hybrid_xc_coeff`) of 0.5 for the short-range exchange. (The default for `hybrid_xc_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.
- `pbesol0` : Hybrid functional in analogy to PBE0 [1], except that the PBEsol [187] GGA functionals are used, mixing 75 % GGA exchange with 25 % Hartree-Fock exchange.
- `b86bpbe-25` : B86bPBE hybrid functional [19, 181], mixing 75 % GGA exchange with 25 % Hartree-Fock exchange.
- `b86bpbe-50` : B86bPBE hybrid functional [19, 181], mixing 50 % GGA exchange with 50 % Hartree-Fock exchange.
- `lc_wpbeh` : Range separated hybrid functional LC-ωPBEh using 100 % Hartree-Fock exchange in the long-range part and ωPBE [179] in the short-range part. The full correlation energy is treated as in PBE. The `hse_unit` must be specified as in hse06!
  Syntax:
  `xc lc_wpbeh` ω α

$$E_{xc}^{\text{LC-}\omega\text{PBEh}} = \alpha E_{xx}^{\text{SR}} + (1-\alpha) E_{x_{\omega\text{PBE}}}^{\text{SR}} + \left(\frac{1}{\epsilon}\right) E_{xx}^{\text{LR}} + \left(1 - \frac{1}{\epsilon}\right) E_{\omega\text{PBE}}^{\text{LR}} + E_{c_{\text{PBE}}} \tag{3.6}$$

  * ϵ can be the dielectric constant. The default value is 1. One might change this parameter with the keyword `lc_dielectric_constant`
  * If $\alpha = 0$ the functional is also known as LC-ωPBE [76]
  * $\alpha = 1$ would correspond to a PBE0 calcuation with 100 % Hartree-Fock exchange

- Hybrid Meta-generalized gradient functionals (including non-local exchange): *Please also see Secs. 3.23 and 3.24 for related keywords and technical hints. Currently the non-local exchange contribution is fixed in all implementations due to the parameterised nature of these density functionals.*

  - `m06` : Truhlar's optimized hybrid meta-GGA of the "M06" suite of functionals; with 27% exact exchange. [249]

  - `m06-2x` : Truhlar's optimized hybrid meta-GGA of the "M06" suite of functionals, with double contribution (54%) from the hartree-fock exact exchange. [249]
  - `m06-hf` : Truhlar's optimized hybrid meta-GGA of the "M06" suite of functionals, with 100% exact exchage contribution. [248]
  - `m08-hx` : Truhlar's optimized hybrid meta-GGA of the "M08" suite of functionals, with 52.23% contribution from the hartree-fock exact exchange. [251]
  - `m08-so` : Truhlar's optimized hybrid meta-GGA of the "M08" suite of functionals, with 56.79% contribution from the hartree-fock exact exchange. [251]
  - `m11` : Truhlar's optimized range-separated local meta-GGA of the "M11" suite of functionals [190]. The range-separation variable is also hardcoded in this implementation with $\omega = 0.25$ bohr^{-1}.

- A substantial further range of functionals is available through the `LibXC` library [151], of which v4.0.2 is distributed with FHI-aims. The implementation covers spin paired and polarised calculations for LDA, GGA and meta-GGA approaches (excluding those meta-GGAs on the laplacian of the density), as well as hybrid-DFT, in equivalence to functionality with canonical functionals (energy, forces and stress). The general syntax is `xc libxc <name>` where `<name>` is using the `LibXC` nomenclature (available at https://www.tddft.org/programs/libxc/functionals/), and exchange and correlation density functionals can be combined using a "+" sign. As an example, the LibXC call for PBE would be `xc libxc GGA_C_PBE+GGA_X_PBE`.

- Alternative implementations of some XC functionals via the `dfauto` program [217]. These implementations are generated automatically from Maple definitions that are located in `xc_dfauto/`. The general syntax is `xc dfauto <name>` where `<name>` can be one of (case-insensitive):

  - `dfauto pw-lda|pbe|pbe0|tpss` : This is practically identical to specifying directly `xc <name>`, and essentially provides alternative implementations of those functionals for testing purposes.
  - `dfauto scan` : This is the meta-GGA functional SCAN [219].
  - `dfauto rscan` : This is a Revision of the meta-GGA functional SCAN (rSCAN) [16].
  - `dfauto scan0` : This is the meta-GGA hybrid functional SCAN0 [114], which mixes SCAN with 25% of exact exchange.

- Double-hybrid functionals (including non-local exchange and correlation): Double-hybrid functionals are emerging quickly in the last decade. "*double-hybrid*" here means that the exchange functional mixes LDA(and/or GGA) exchange with "Hartree-Fock like exact exchange". Meanwhile, the correlation functional is composed of both conventional LDA(and/or GGA) correlation and second-order perturbation energy. Doubly-hybrid functionals are "*semi-empirical*", generally including several empirical parameters determined by optimizing against one or several well-chosen

databases. Double-hybrid functionals show a remarkable improvement over conventional (hybrid-)GGAs in the description of heats of formation, bond dissociation enthalpies, reaction barrier heights and weak interactions of the main group elements. Doulbe-hybrid functionals have become new leading actors in the field of computational chemistry.

  - `xyg3` : Double-hybrid functional XYG3, containing 80.33% Hartree-Fock exchange and 32.11% second-order perturbation energy [246].
  - `xdh-pbe0` : Double-hybrid functional xDH-PBE0, containing 83.51% Hartree-Fock exchange and 52.42% opposite-spin second-order perturbation correlation [245].

- Some specific correlated methods: *Only a subset. For many correlated methods that can be used as non-selfconsistent perturbative post-processing methods after an initial s.c.f. calculation, see the* `total_energy_method` *keyword. Most of these are not available for periodic geometries, or, if at all, in a very experimental state.*

  - `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 [86], 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. *Note added in March 2016: A periodic implementation of MP2 is available but, at the time of writing, computationally extremely expensive. If you decide to use it, please keep in mind that the periodic version is included here as a matter of protocol but is not yet optimized to be fully usable in production calculations.*

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

$$\frac{1}{r-r'} \rightarrow \frac{1}{\varepsilon(r,r')} \cdot \frac{1}{r-r'} \tag{3.7}$$

    $\varepsilon(r,r')$ is the non-local *microscopic* dielectric function, obtained in the $\omega \rightarrow 0$ frequency limit of the random-phase approximation (RPA). See Ref. [97] for details.

  - `cohsex` : *experimental!* Self-consistent screened exchange plus Coulomb-hole (COH) correlation. See Ref. [97] for details.

- Method of non-local correlation using the "van der Waals density functional" (vdw-DF) as presented by Dion and coworkers in Ref. [59]. Two options are available for the exchange part:

  - `pbe_vdw` : the functional with pbe exchange

– `revpbe_vdw` : the functional with revpbe exchange

Note that this keyword is **not** the correction due to Tkatchenko and Scheffler 2009 [224]. To activate the Tkatchenko-Scheffler correction instead, use the `vdw_correction_hirshfeld` keyword. The functional by Dion *et al.* is a very different functional. As implemented here, it is also *much* more expensive than the Tkatchenko-Scheffler correction. To use the functional by Dion *et al.*, please review the numerical options described in Sec. 3.22.2.

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. [32, 2, 230, 65] and others). While our default settings should be safe, you may wish to consult Sec. 3.23 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. [95], 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. [28], the basis functions of FHI-aims take the format

$$\phi(\boldsymbol{r}) = u(r)/r \cdot Y_{lm}(\theta, \phi) \tag{3.8}$$

in spherical coordinates (r, θ, ϕ) 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.

In addition to our own case studies,[28] the accuracy of the "tier" (sometimes called "FHI-aims-2009"[244]) preconstructed basis sets for semilocal and hybrid DFT calculations was established in several benchmark assessments.[152, 121]

For beyond-DFT methods like MP2, GW, the random-phase approximation, etc., basis set convergence is very different and absolute convergence is often not possible. In the case of beyond-DFT methods, we recommend to ascertain basis set convergence by performing specific convergence tests for any important results. For total-energy

differences between different structures, a counterpoise correction can often be employed (using the `empty` keyword to create sites which have basis functions but no atoms). For beyond-DFT calculations for light elements (H-Ar), the NAO-VCC-nZ basis sets[244] are additionally available among the species defaults and provide reliable convergence for total-energy methods (see Ref. [244] for details).

Finally, we can also use different approaches to create the free-atom-like core and valence basis functions that are included in the so-called "minimal basis" of the NAO basis sets provided with FHI-aims (unless explicitly excluded using the `include_min_basis` keyword). If semilocal density functionals are used in the overall calculation, these atomic radial functions are created for the same exchange-correlation functional as requested by the `xc` keyword of the `control.in` file. For hybrid density functionals, FHI-aims has historically used LDA- or GGA-derived free-atom-like basis functions by default (still current as of August 2017, but slated to be changed in the future), but the definition of the "minimal basis" can be changed in a limited way using the keywords `atomic_solver_xc` or `atomic_solver` . See, for instance, Figure 7 in Ref. [196] for the effect of changing the mimimal basis definition by keyword `atomic_solver_xc` on total energy convergence for a simple example.

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: `atomic_solver` (control.in)

Usage: `atomic_solver string`

Purpose: Changes the atomic solver library that generates the free-atom-like radial functions (or free-ion-like radial functions) used, e.g., in the minimal basis part of the NAO basis sets.

`string` is the name of the solver to be used, either `sratom` or `atom_sphere`. Default: `sratom`

The definition of free atom radial functions used as the "minimal basis" of FHI-aims affects the absolute convergence of total energies calculated by FHI-aims. The radial shape of the free-atom core and valence functions towards the nucleus is nearly exact also for bonded structures if the same exchange-correlation functional is used and thus, e.g., using DFT-PBE generated radial functions when using `xc pbe` will improve the convergence of total energies. (Other quantities, such as atomization energies or other energy differences, will often exhibit better convergence than the total energy, since the effect of the exact shape of the minimal basis functions near the nucleus often cancels to a reasonable extent. See, e.g., Ref. [121] for a study of these effects for Gaussian-type and NAO basis sets compared to accurate reference values.)

Options for `string`:

`sratom`: FHI-aims' default solver for the electronic structure of spherical free atoms on a dense logarithmic grid, called "sratom" (for "scalar relativistic atom") is the same solver as used in the Fritz Haber Institute 1998 pseudopotential generation code by Martin Fuchs and coworkers, reference [75]. It was modified by Timo Jacob to incorporate ZORA scalar relativity when needed. This solver can handle semilocal density functionals but not exact exchange. Thus, for hybrid density functionals in FHI-aims, the "minimal basis" radial functions produced by sratom are semilocal DFT, not hybrid functional basis functions. This leads to a slower convergence of absolute total energies with hybrid functionals in FHI-aims (the error in atomization energies cancels to a good extent). See Figure 7 in Ref. [196] for the magnitude of this effect for the example of the convergence of Hartree-Fock calculations for Au_2 with an LDA-generated minimal basis compared to the same calculation, but with a minimal basis derived using the Krieger-Li-Iafrate (KLI) approximation [138, 139, 140] to the exact-exchange optimized effective potential. "sratom" is the current default solver for radial functions in FHI-aims.

`atom_sphere`: This solver for spherical free atoms was developed in Stefan Goedecker's group for many years (beginning with Ref. [78]) and includes support for semilocal and hybrid density functionals. In FHI-aims, "atom_sphere" is used as a library under the Lesser General Public License (LGPL). As of this writing (August 2017), only non-relativistic calculations are supported by "atom_sphere". In this case, the resulting total energies for hybrid density functionals converge precisely as well as their semilocal DFT equivalents, as shown, e.g., in Ref. [121]. "atom_sphere" will only work if support for libxc was compiled into the FHI-aims binary used.

Tag: `atomic_solver_xc` (control.in)

Usage: `atomic_solver_xc string`

Purpose: Changes the exchange-correlation functional used to generate the free-atom-like radial functions in the minimal basis part of the NAO basis sets.

Restriction: This keyword only has an effect if `atomic_solver sratom` is used.

`string` is the name of the exchange-correlation functional to be used for the free-atom solution. Default: Internal defaults (no specific option).

One specific option for this keyword is:

`KLI`

for the Krieger-Li-Iafrate (KLI) approximation [138, 139, 140] to the exact-exchange optimized effective potential. Figure 7 in Ref. [196] is an example of the effect of the choice of different minimal basis functions on the convergence of the total energy of a Hartree-Fock calculation for a heavy element. In that case, the KLI approximation is closer to the converged result since the radial behavior of the minimal basis functions towards the nucleus is closer to the Hartree-Fock result, and because the radial function behavior of the free atom near the nucleus is nearly identical to the near-nuclear behavior of the Kohn-Sham orbitals in the bonded structures. In other words, for Hartree-Fock, KLI represents the nuclear cusp better than the local-density approximation (which is the present default choice of minimal basis radial functions for Hartree-Fock).

In addition, the following functionals have been implemented for the atomic solver and may be specified even if a different functional is chosen for the `xc` keyword:

`pz-lda, pw-lda, vwn, vwn-gauss, pw91_gga, pbe, rpbe, revpbe, pbeint, blyp`

In all cases where `atomic_solver_xc` is not specified, the choice of the exchange-correlation functional for free atoms varies with the chosen setting for the `xc` keyword, but is not necessarily identical. For local and semilocal density functionals, the choice of functional for the minimal basis is generally identical to keyword `xc`, but for hybrid functionals, the choice may vary and is currently "best" documented in the source code (in subroutine get_free_atoms.f90).

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.`

This keyword automates a specialized version of the counterpoise correction and should only be used with great care. A general counterpoise correction for molecules can be implemented manually and in separate steps using the `empty` keyword. The

atomization BSSE correction implemented above, if used, must be checked very carefully to ensure that all single-atom reference calculations carried out in the process reached the exact same atomic reference state. Many important atoms have more than one self-consistent solution, and mixing different self-consistent solutions may produce completely erratic results. Our recommendation therefore to carry out any counterpoise correction manually instead, by doing separate single-point FHI-aims calculations with different molecular fragments and different basis definitions.

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

$$\Delta_{ac} = \sum_x [E^x(x) - E^x(sys)] \tag{3.9}$$

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

$$E^{BSSE} = E^{sys}(sys) + \Delta_{ac} \tag{3.10}$$

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

$$E_{at}^{BSSE} = E^{sys}(sys) - \sum_x E^x(sys) \tag{3.11}$$

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.

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

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_vector` s are listed in `geometry.in`, through the definition and order of the reciprocal lattice. By default, the resulting `k_grid` is centered around the Γ-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_grid_density` (control.in)

Usage: `k_grid_density` `density`

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

`density` : float number in units of $1/\text{Å}^{-1}$ defining the density of k-point splits along the reciprocal axis of the first Brillouin zone

---

Tag: `k_offset` (control.in)

Usage: `k_offset` `f1 f2 f3`

Purpose: Defines a possible non-Γ 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-Γ for better k-space sampling. For example, (0.5, 0.5, 0.5) together with even $\mathtt{n}i$ for `k_grid` defines a Monkhorst-Pack [170] type grid. See Sec. 4.3 for details.

The `k_offset` cannot be used with Hartree-Fock or hybrid density functionals as the current implementation assumes a Γ-centered grid.

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 [47], etc.

The k_points_external keyword cannot presently be used with Hartree-Fock or hybrid density functionals as the current implementation the presence of the k_grid keyword and information in control.in.

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. During s.c.f., these values are for reference only. However, they may be used for postprocessing after the s.c.f. cycle is complete, e.g., for keyword output postscf_eigenvalues , where the external k-point list k_list.in is not supported.
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: onsite_accuracy_threshold (control.in)

Usage: onsite_accuracy_threshold threshold

Purpose: Issues a warning if the onsite integral of a basis function on the 'radial' integration grid differs from the same onsite integral on the more accurate logarithmic grid by more than threshold.

threshold is given in eV. Default: 0.03 eV.

See keyword output onsite_integrands for the integrals that are actually evaluated.

For "light" settings, you can most likely ignore the associated warning. Possibly, treat it as a reminder to check final results with "tight" settings or similar, just in case.

FHI-aims writes the respective onsite integral values for all its radial functions side by side after the setup of all radial functions is complete. Too large deviations between the calculated values on the radial and logarithmic grids can indicate accuracy problems, which is a particular concern for high-accuracy benchmark calculations. In particular, Gaussian-type orbital basis sets of the Dunning type that are used for high-level reference calculations need grids that are far more accurate than normal NAO basis sets, due to the unphysical wiggles that contracted Gaussian functions with high exponents introduce near the nucleus.

If the associated warning strikes, the grid accuracy can be improved either using the radial keyword or (simpler) the radial_multiplier keyword.

However, if onsite_accuracy_threshold triggers a warning for "light" settings, most likely you can safely ignore the warning. With light settings, the point may be to do a reduced-accuracy calculation, which should still be safe for its original purposes. Just check the radial grid when in doubt.

If the flag override_integration_accuracy is toggled to .false., however, FHI-aims does stop whenever it encounters a radial function whose onsite integrals are not deemed accurate enough by the onsite_accuracy_threshold criterion.

Tag: `override_integration_accuracy` (control.in)

Usage: override_integration_accuracy flag

Purpose: If set to false, FHI-aims stops calculations for which the onsite integral of any radial function is less accurate than prescribed by keyword onsite_accuracy_threshold .

flag can be .true. or .false. Default: .true.

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}. Lowered to 10^{-8} by default if output i s requested.

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.

For electron densities or orbitals plotted for visualization using cube files (output f unctionality), a too high value of wave_threshold can sometimes lead to small but visible discontinuities. Thus, the default threshold is lowered to $10^{-}8$ if output i s requested.

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_{onset} is smaller that `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 ($v_c(r)$=∞ 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 $v_c(r)$=∞ for $r > r_{\text{cut}} = r_{\text{onset}} + w$, where r_{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. [125])

- `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^{-2}].
`alpha_i coeff_i` : *If* `N`>1, $i = 1, \ldots, N$ additional lines specify exponents α_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,

$$u(r) = \frac{1}{Norm} r^{L+1} \cdot \exp(-\alpha r^2), \tag{3.12}$$

or *non-primitive* linear combinations.

$$u(r) = \frac{1}{Norm} r^{L+1} \cdot \sum_{i=1}^{i=N} g_i \exp(-\alpha_i r^2). \tag{3.13}$$

In existing quantum chemistry codes, Gaussian basis functions can be defined either as *spherical* Gaussians [Eq. (3.12) above], or as *cartesian* Gaussians,

$$\phi(\boldsymbol{r}) = x^k y^m z^n \exp(-\alpha r^2), \quad \text{where} \quad k + m + n = L. \tag{3.14}$$

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_{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 numerically tabulated free-atom basis functions (core and valence) 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.

With Gaussian basis sets and any other basis sets that should not include any radial functions of the numeric atom-centered orbital spherical free atom, `include_min_basis` must be set to `.false.`

If `include_min_basis` is set to `.true.`, the basis set *will* additionally include the numerical spherical free-atom radial functions. For our standard FHI-aims numeric atom-centered orbital basis sets, this behavior is desired and part of their definition. However, if these basis functions are added to a standard Gaussian-type basis set, then total energies and other computed quantities will not be the same as, say, with a pure Gaussian-type orbital code.

Note that, due to the presence of the free-atom radial functions, our numeric atom-centered basis sets usually give lower total energies for DFT than standard Gaussian basis sets, because the description of the region near the nucleus is more precise. This is demonstrated quantitatively, including figures, in the appendix of Ref. [244].

`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.

---

species sub-tag: sto (control.in)

Usage: `sto n l zeta`

Purpose: Adds a Slater-type orbital to the basis set.

Notes: This basis function type is not subject to a cutoff potential.

`n` is an integer which plays the role of the principal quantum number for the STO
`l` is an integer specifying the STO angular momentum
`zeta` is a double precision number specifying the STO exponent, which plays the role of the effective nuclear charge

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. [28, 96], 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. [96] for details regarding integration batches.

Tag: `force_lebedev` (control.in)

Usage: `force_lebedev type`

Purpose: Allows to switch between Delley's [57] angular grids (17 digits) and the original angular grids tabulated by Lebedev and Laikov [147, 148, 149] (12 digits). And also, the ESTD and D6h grid are supported here.

`type` is a keyword (string), either `original` or `Delley` or `estd` or `d6hgrid`. 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, charactrizing 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. [96]. In particular, we support:

- maxmin : The default for serial computations: the "grid-adapted cut-plane" method described in Ref. [96]
- 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. [96]
- parallel_octree Parallel version of the "octree" method described in Ref. [96]. 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 arround 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. [96], 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. [28] for details regarding "partitioning of unity" of the charge density in integrations and the Hartree potential. The partition functions $p_{\mathrm{at}}(\boldsymbol{r})$ are only calculated if their denominator (the norm; e.g. $\sum_{\mathrm{at}} \rho_{\mathrm{at}}/|\boldsymbol{r} - \boldsymbol{R}_{\mathrm{at}}|^2$ 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_sparse`

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:

$$p_{\mathrm{at}}(\boldsymbol{r}) = \frac{g_{\mathrm{at}}(\boldsymbol{r})}{\sum_{\mathrm{at}'} g_{\mathrm{at}'}(\boldsymbol{r})} \,. \tag{3.15}$$

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

- `rho_r2` : $g_{\mathrm{at}}(\boldsymbol{r}) = n_{\mathrm{at}}^{\mathrm{free}}(r)/r^2$
  (first suggested by Delley [56]).
- `rho_r` : $g_{\mathrm{at}}(\boldsymbol{r}) = n_{\mathrm{at}}^{\mathrm{free}}(r)/r$
- `rho` : $g_{\mathrm{at}}(\boldsymbol{r}) = n_{\mathrm{at}}^{\mathrm{free}}(r)$
- `fermi` : *Deprecated—do not use.* A Fermi-function like approach, requires two additional parameters.

- `stratmann`: The shape suggested by Stratmann *et al.*, Ref. [218]. This saves ≈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. [218]. 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 μ_{ik} in Eq. (4) of Ref. [218] 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. [218] 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.

- `stratmann_sparse`: This version of the `Stratmann` partition table is the same as `stratmann_smoother`, but it stores the relevant interatomic distances in a memory saving form.

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 for most calculations. When GPU acceleration is used for tasks involving the batch integration scheme, this value is raised to 200.

See `grid_partitioning_method` and Ref. [96] 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 angu-

lar 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 and for RI_method LVL; infinite for Hartree-Fock, hybrid functionals, GW, etc. if RI_method V is used.
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_{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 α 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. [14]

`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_{outer}, in Å.

The location of the `number` radial shells is given by

$$r(i) = r_{\text{outer}} \cdot \frac{\log\{1 - [i/(N+1)]^2\}}{\log\{1 - [N/(N+1)]^2\}} \tag{3.16}$$

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,∞].

The FHI-aims `species_default` files provide values for `number` according to the formula N=1.2×14(`nucleus` +2)$^{1/3}$, as determined empirically in Ref. [14]. 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 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 .

The effect of the radial_multiplier is explained in Ref. [244] (open access at http://iopscience.iop.org/1367-2630/15/12/123033/article) Look at Figure A.1 and the accompanying explanation in that reference.

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. [28] 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 ≈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. [28].

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. [28], our experience is that energy differences are usually well converged for l_{hartree}=4, and total energy convergence at the level of a few meV is reached at l_{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, $\sum_{\text{at}} n_{\text{at}}^{\text{free}}(|\boldsymbol{r} - \boldsymbol{R}_{\text{at}}|)$:

$$v^{\text{es,free}}(\boldsymbol{r}) = \sum_{\text{at}} v_{\text{at}}^{\text{es,free}}(|\boldsymbol{r} - \boldsymbol{R}_{\text{at}}|) \tag{3.17}$$

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

$$\delta n(\boldsymbol{r}) = n(r) - \sum_{\text{at}} n_{\text{at}}^{\text{free}}(|\boldsymbol{r} - \boldsymbol{R}_{\text{at}}|) \tag{3.18}$$

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

$$\delta \tilde{n}_{\text{at},lm}(r) = \int_{r=|\boldsymbol{r}-\boldsymbol{R}_{\text{at}}|} d^2\Omega_{\text{at}} p_{\text{at}}(\boldsymbol{r}) \cdot \delta n(\boldsymbol{r}) \cdot Y_{lm}(\Omega_{\text{at}}) \tag{3.19}$$

(the sum of all partition functions at every point is always unity). Due to the finite extent of $\delta n(\boldsymbol{r})$ and $p_{\text{at}}(\boldsymbol{r})$ (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

$$\delta \tilde{v}_{\text{at},lm}(r) = m_{\text{at},lm}/r^{l+1} \quad . \tag{3.20}$$

Instead of a spline evaluation, faraway atoms can thus be analytically accounted for using tabulated, constant *multipole moments* $m_{\mathrm{at},lm}$. 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,

$$v_{\mathrm{es}}(\boldsymbol{r}) = v^{\mathrm{es,free}}(\boldsymbol{r}) + \sum_{\mathrm{at},lm} \delta\tilde{v}_{\mathrm{at},lm}(|\boldsymbol{r} - \boldsymbol{R}_{\mathrm{at}}|)Y_{lm}(\Omega_{\mathrm{at}}) \quad . \tag{3.21}$$

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 $\delta\tilde{n}_{\mathrm{at},lm}(r)$ are here additionally split into a short-ranged real-space part, and a smooth, long-ranged reciprocal-space part (Ewald's method), by splitting

$$\frac{1}{r} = \frac{\mathrm{erf}(r/r_0) + \mathrm{erfc}(r/r_0)}{r} \tag{3.22}$$

(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 parameter r_0 can be very important to determine the efficiency of the actual evaluation of the Hartree potential in periodic systems; it can be set in `control.in` using the `Ewald_radius` keyword. The keyword is adaptive to some extent but especially for slab systems or 2D systems with large vacuum regions, specifying the value of `Ewald_radius` by hand can lead to significant performance improvements. (FHI-aims can accommodate very large vacuum regions, e.g., 100 Å, efficiently if this parameter is set correctly.)

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

$$|\boldsymbol{G}_{\mathrm{max}}|^{l^{\mathrm{es}}_{\mathrm{max}}-2} \cdot \frac{1}{\boldsymbol{G}^2_{\mathrm{max}}} \cdot \exp(-\frac{r_0^2\boldsymbol{G}^2_{\mathrm{max}}}{4}) = \frac{\eta}{10 \cdot 4\pi} \, . \tag{3.23}$$

Our default choice for η (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. (3.23) is slightly modified compared to the version given in Ref. [28].

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 calculting

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 suffiently coarse, time is saved because of the reduced number of potential computations.

We use an envenly 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 $\Omega(r) = \bar{q}\,\mathrm{erf}(r/r_0)/r$ 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 $\delta\tilde{v}_{\mathrm{at},lm}(r)$ 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 $\delta\tilde{v}_{\mathrm{at},lm}(r)$ of the real-space (Ewald!) Hartree potential are not evaluated for distances where $\delta\tilde{v}_{\mathrm{at},lm}(r) <$`threshold`. This tag provides similar functionality as the `multipole_threshold` tag for the cluster case (numerically different due to the absence of $\mathrm{erf}(r/r_0)$ in the cluster case).

Tag: `compensate_multipole_errors` (control.in)

Usage: `compensate_multipole_errors flag`

Purpose: If true, introduces a compensating normalization and density to eliminate the effects of small charge integration errors in the long-range Hartree potential.

`flag` is either `.true.` or `.false.`. Default: `.true.` `.false.` only if `DFPT_phonon` or `magnetic_response` is requested.

This keyword is especially useful when assessing the electrostatic potential far away from a structure, e.g., when calculating a surface dipole correction (for asymmetric slabs) or work function. See `use_dipole_correction` or `evaluate_work_function` for details on these methods.

In general, keyword `compensate_multipole_errors` makes sure that the long-range charge components of the Hartree potential are exactly those expected from the calculated (and normalized) electron density. Any small spurious non-zero components that are solely due to integration errors on a finite integration grid.

Tag: `Ewald_radius` (control.in)

Usage: `Ewald_radius value`

Purpose: Governs the Ewald-type short-range / long-range splitting of the Coulomb potential in Eq. (3.22).

`value` : Either a string `automatic`, or the range separation parameter r_0 in Eq. (3.22) (in bohr). Default: `automatic`.

May also be specified as `hartree_convergence_parameter` or `ewald_radius` .

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.)

This parameter is performance critical especially for slab calculations (2D material or surface) with a large vacuum region.

If the string `automatic` is chosen, then the parameter r_0 is set according to an empirically determined function as follows:

$$r_0 = A_0 \cdot (v - A_1)^{1/3} \quad , \tag{3.24}$$

subject to the limiting conditions 2.5 bohr$\leq r_0 \leq$5.0 bohr. Here, v is the specific volume (unit cell volume divided by number of atoms), and A_0=1.47941 bohr and A_1=1.85873 Å^3 are empirically determined parameters.

The chosen empirical form was tested and adapted for a slab model of a 2D material with a vacuum region up to 50 Å. For such systems, this choice entails a significant performance improvement; and for larger vacuum regions, even larger choices than r_0=5.0 bohr are possible. However, the same empirical relation may not be optimal for moderately dense solids (such as GaAs), where smaller choices of r_0 can perform better. Overall, the optimum choice of r_0 would be to adapt it on the fly over the course of a given calculation, but implementing such an adaptive algorithm has not yet been done.

For a yet more refined choice, further testing would be necessary, as well as a dependence on `hartree_fourier_part_th` (which is not yet incorporated).

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

Usage: `ewald_radius value`

Purpose: Governs the Ewald-type short-range / long-range splitting of the Coulomb potential in Eq. (3.22).

`value` : Either a string `automatic`, or the range separation parameter r_0 in Eq. (3.22) (in bohr). Default: `automatic`.

This keyword has exactly the same meaning as the `Ewald_radius` kewyord.

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. (3.22).

`value` : Either a string `automatic`, or the range separation parameter r_0 in Eq. (3.22) (in bohr). Default: `automatic`.

This keyword has exactly the same meaning as the `Ewald_radius` kewyord.

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 $|\boldsymbol{G}_{\mathrm{max}}|$ for the Fourier summation of the long-range electrostatic potential (Ewald).

threshold is a real positive small number [η in Eq. (3.23)]. Default: $5{\cdot}10^{-7}$ (in atomic units) .

See Eq. (3.23). 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 $p_{\mathrm{at}}(\boldsymbol{r})$ is used to split $\delta n(\boldsymbol{r})$ 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_sparse

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 $\mathrm{erf}(r_{\mathrm{outer}}/r_0) < \texttt{threshold}$. This is then used to determine which atoms need be included in the second term (sum over atoms) of Eq. (3.21).

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. (3.20), the multipole moments $m_{\mathrm{at},lm}$ are determined by an explicit integration of the finite real-space density component $\delta\tilde{n}_{\mathrm{at},lm}(r)$. However, for very high l, even spuriously small density components (10^{-10} or lower) may be artificially weighted up in $m_{\mathrm{at},lm}$; on a finite integration grid, $m_{\mathrm{at},lm}$ 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 $m_{\mathrm{at},lm}/r_{\mathrm{mp}} <$`threshold` for all $l \geq l_{\mathrm{thr}}$, 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 $\delta\tilde{n}_{\mathrm{at},lm}(r)$ for the purpose of determining the far-field moments $m_{\mathrm{at},lm}$.

`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 $|\delta\tilde{n}_{\mathrm{at},lm}(r)| <$`threshold`. The actual $m_{\mathrm{at},lm}$ 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 $n_{\mathrm{at}}^{\mathrm{free}}$

`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 $n_{\mathrm{at}}^{\mathrm{free}}(r)$ 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 $m_{\mathrm{at},lm}$ are used to construct the Hartree potential $v_{\mathrm{es}}(\boldsymbol{r})$

`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* $\delta\tilde{n}_{\mathrm{at},lm}(r) <$`threshold` for a given atom. At a given integration point $\boldsymbol{r}$, $v_{\mathrm{es}}(\boldsymbol{r})$ is assembled by evaluating Eq. (3.21). 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 $\delta\tilde{v}_{\mathrm{at},lm}(r)$ 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 $\delta\tilde{v}_{\mathrm{at},lm}(r)$ are not evaluated for distances where $\delta\tilde{v}_{\mathrm{at},lm}(r) <$`threshold`.

This tag provides similar functionality as the adaptive_hartree_radius_th tag for the periodic case (numerically different due to the absence of $\mathrm{erf}(r/r_0)$ in the cluster case).

Tag: `normalize_initial_density` (control.in)

Usage: `normalize_initial_density flag`

Purpose: If true, normalizes the initial electron density to reproduce the exact intended number of electrons when integrated on the three-dimensional, overlapping atom-centered integration grid of FHI-aims.

`flag` is either `.true.` or `.false.`. Default: `.true.`

This keyword only normalizes the initial density. It should always be an exact subset of the functionality provided by compensate_multipole_errors . If used in conjunction with collinear spin and a geometry optimization (or sc_init_iter), no subsequent renormalizations are performed, except for runs which use a fixed_spin_moment .

The default for normalize_initial_density was set to `.false.` before August, 2017.

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.

If use_dipole_correction or evaluate_work_function are specified, omitting the keyword set_vacuum_level causes FHI-aims to automatically determine a suitable z.

However, a vacuum plane will only be determined if the nearest atom is more than 6 Å away from the vacuum level. Determining a surface dipole for distances for which basis functions or charge densities could overlap might lead to errors. Since FHI-aims does allow one to use rather large vacuum spacings at low (if any) computational overhead, the calculation will stop for too small vacuum spacings and alert the user.

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.

However, it must be possible to find a vacuum plane z, where the surface dipole is compensated, that is further than 6 Å away from the nearest atom. Otherwise, the calculation will stop and alert the user.

The reason is that a surface dipole cannot be safely determined for vacuum spacings for which basis functions or charge densities could overlap. This can lead to errors. Note that FHI-aims does allow one to use rather large vacuum spacings at low (if any) computational overhead.

Attention: This keyword is currently disabled for charged periodic systems. The Coulomb potential of a charged surface slab will reach far into the vacuum, apparently 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 user to 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. Charged periodic calculations with a vacuum region are physically questionable for very different reasons in any case; we recommend to find a different workaround with explicit charges whenever that is possible.

Note that for very large surface slabs, this keyword might cause instabilities in the SCF cycle. If you suspect this to be the case and remove `use_dipole_correction` from your `control.in`

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 explicitely.

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 $\delta n(\boldsymbol{r})$ into $\delta\tilde{n}_{\mathrm{at},lm}(r)$ for the present species. *Must be specified.*

As pointed out in Ref. [28], our experience is that energy differences are usually well converged for l_{hartree}=4, and total energy convergence at the level of a few meV is reached at l_{hartree}=6. Only in exceptional cases should different settings be required.

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

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. 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 primarily intended to obtain qualitatively accurate relativistic corrections for energy band structures and eigenfunctions. A detailed benchmark of the accuracy of the approach is given in Ref. [113]. *Importantly, changes to the total energy beyond the sum-of-eigenvalues are not included, and total energy gradients (forces) are also unavailable.*
- An essentially fully relativistic treatment of the Kohn-Sham kinetic energy – specifically, the so-called quasi-four-component (Q4C) approximation – is nearing completion. In physical terms, this includes self-consistent spin-orbit coupling as well as the so-called "small component" of the Dirac eigenfunctions, where the small component is based on a free-atom-like approximation. At the time of writing (March 2020), the Q4C approach is still restricted to total energies and Kohn-Sham eigenvalues for closed-shell systems at the level of semilocal DFT. *The Q4C implementation is still considered highly experimental and therefore not yet fully documented. In particular, parallelization is not yet complete, forces are not available and support for hybrid DFT is not yet available. Please do not use the approach without contacting Rundong Zhao and Volker Blum.*

Scalar relativity and spin-orbit coupling

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.

The recommended level of scalar relativity in FHI-aims is the so-called "atomic ZORA" approximation, as defined specifically in Equations (55) and (56) of Ref. [28]. It is

important to refer to this specific definition since there are other variants of ZORA ("zero-order regular approximation") in the literature and in other codes, including variants also called "atomic ZORA" but following a different mathematical definition.

The keyword needed to use this level of theory is

`relativistic` atomic_zora scalar

That's it.

The "atomic ZORA" level of theory as implemented in FHI-aims has held up extremely well in large, high-accuracy benchmarks of scalar-relativistic total-energy based properties [152] as well as energy band structures [113]. It works for the right mathematical reasons. It can, in principle, be used across the entire periodic table (there should be no need to resort to non-relativistic calculations except for benchmarking purposes).

In addition, a non-selfconsistent treatment of spin-orbit coupling for band structures, densities of states, absorption properties and for the independent-particle dielectric response is also available and can be used in addition to (on top of) scalar relativistic calculations using the atomic ZORA. This is described in detail in Ref. [113], including a simple discussion of relativistic treatments in general and of how the "atomic ZORA" and the spin-orbit coupling formalism on top of it are related.

The keyword to add post-scf spin-orbit coupling is

`include_spin_orbit`

That's it. Note that this keyword can be used as a followup to both non-spinpolarized and spin-polarized scalar-relativistic calculations.

An important fact to keep in mind is that a scalar-relativistic calculation yields two distinct spin sets of spin states, one for each spin channel. However, after the perturbative spin-orbit coupling treatment, only a single set of states emerges as output, since spin-orbit coupling mixes the scalar-relativistic spin states and the spin channels are no longer distinct. Thus, the output files for any quantities derived from spin-orbit coupled calculations (densities of states, band structures, etc.) are not and cannot be printed as separate spin channels - only one set of files is written that includes states derived fron both former spin channels.

More details follow below, but here are three additional important point:

- Never mix results from different scalar relativistic treatments in total-energy differences. Absolute total-energy differences between different relativistic treatments can be very large because the deep core state energies change.

- The absolute core level energies in the "atomic ZORA" approximation are far away from measured core level energies that would appear in experiment or in the actual Dirac equation. However, the relative core level shifts (differences) between different chemical systems are still reliable.

- FHI-aims also includes another relativistic treatment called "scaled ZORA" but this seems to be slightly less accurate and does not have support for forces or

stresses or any other use cases. We do not recommend to use "scaled ZORA" any more ("atomic ZORA" simply seems to do the better job).

More details on spin-orbit coupling

Spin-orbit coupling (SOC) is a simple consequence of transforming Dirac's equation to a (two-component) Schrödinger-like form. This leads to an approximate "spin-orbit-coupled" Hamiltonian of the form

$$\hat{H} = \hat{t}_{SR} + \hat{v} + \hat{v}_{SOC}, \tag{3.25}$$

where $\hat{t}_{SR}$ is the usual scalar relativistic kinetic energy operator (e.g., atomic ZORA), $\hat{v}$ is the local or non-local potential, and $\hat{v}_{SOC}$ is the spin-orbit coupling operator,

$$v_{SOC} = \frac{i}{4c^2}\boldsymbol{\sigma} \cdot \boldsymbol{p}v \times \boldsymbol{p}. \tag{3.26}$$

FHI-aims currently implements a treatment of spin-orbit coupling which adds spin-orbit coupling corrections to the Kohn-Sham eigenvalues, band structures, and densities of states in a single evaluation *after* the scalar-relativistic s.c.f. cycle has converged. This means that it is a post-processed implementation of spin-orbit coupling. It works in the Hilbert space of calculated scalar-relativistic eigenstates, as opposed to the "full" space spanned by the computational basis set, to dramatically reduce the problem size. This is known as the "second-variational" method. It only calculates and diagonalizes the spin-orbit-coupled Hamiltonian once; therefore, the resulting spin-orbit-coupled eigenstates are non-self-consistent.

Full details on the implementation of spin-orbit coupling in FHI-aims, as well as a derivation of the spin-orbit-coupled Hamiltonian from the Dirac equation and a detailed benchmark of the effect of spin-orbit coupling on band structures, may be found in Ref. [113] When publishing results using spin-orbit coupling in FHI-aims, please remember to cite this reference.

Applying the SOC operator as a correction to scalar-relativistic eigenvectors is quantitatively accurate (to a few 0.01 eV for valence band structures) for elements below Xe (Z=54) when combined with atomic ZORA. *For heavy elements (approximately Au and beyond) this level of theory is only qualitatively accurate. It captures the majority of the SOC effect, but quantitative deviations above 0.1 eV for band structures must be expected. Similarly, the corrections for any core levels would require one to go beyond non-self-consistent SOC.*

Since this implementation of spin-orbit coupling operates in the Hilbert space spanned by the calculated scalar-relativistic eigenvectors, for accurate high-lying bands one must include sufficiently many unoccupied states. This may be done by setting `empty_states` to a higher value or, if you are feeling particularly paranoid, setting the `calculate_all_eigenstates` keyword to include all possible eigenstates. It is the opinion of the authors that this is only relevant for materials containing Au and heavier elements.

Which Parts of FHI-aims Support Spin-Orbit Coupling?

The current spin-orbit coupling implementation started in 2014. FHI-aims has been in development since 2004. While we are actively working on enhancing support for spin-orbit coupling and relativistic schemes beyond throughout FHI-aims , due to the sheer size of the code base some of the functionality in FHI-aims does not have spin-orbit coupling support. Enabling the SOC keyword will do nothing for functionality that has not been modified to support SOC, and the code will return scalar-relativistic values. A **partial** list of functionality that does support SOC and will output spin-orbit-coupled values is

- Band structure calculations
- Densities of state calculations, both interpolated and non-interpolated
- Mulliken analyses
- Atom/species-projected densities of state
- Dielectric functions and absorption coefficients
- Orbital cube plotting

In general, spin-orbit coupling can be applied both for non-spinpolarized and spin-polarized scalar-relativistic input. However, after spin-orbit coupling is applied, only one set of states remains since the spin channels are no longer separated (spin-orbit coupling mixes the formerly separate spin channels).

The best way to determine whether a particular method supports spin-orbit coupling is to look at its manual entry.

One advantage of post-processed SOC is that one still has access to scalar-relativistic values, as the spin-orbit-coupled values are generated from the scalar-relativistic values. Physical insight may be gained by comparing scalar-relativistic and spin-orbit-coupled values against one another. For example, strong spin-orbit splitting of eigenstate is a dead giveaway that it contains p-orbitals for a heavy species. When spin-orbit coupling is enabled, FHI-aims will output both scalar-relativistic and spin-orbit-coupled values whenever this is computationally feasible. For methods supporting spin-orbit coupling that output results to files, the files containing the scalar-relativistic values will have an additional suffix ".no_soc" to distinguish them from the spin-orbit-coupled values.

Another advantage of post-processed SOC is, simply, computational efficiency. Particularly hybrid DFT calculations are already extremely demanding at the non-spin-polarized, scalar-relativistic level of theory. The ability to pursue SOC corrections after, rather than during a self-consistent scalar-relativistic allows us to access significantly larger problem sizes than would otherwise be possible.

Tags for general section of `control.in`:

Tag: `include_spin_orbit` (control.in)

Usage: `include_spin_orbit method`

Purpose: Include the effects of spin-orbit coupling, when supported, in post-processed features of FHI-aims. When using spin-orbit coupling in your calculation, please cite Ref. [113]

`method` The method for including spin-orbit coupling. At present, only `type non_self_consistent` is suitable for production-level calculations.

Note: While FHI-aims also prints out a corrected total-energy expression based on the SOC-corrected eigenvalues, do not use this value. It is experimental.

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. [28]. 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. [28]. `threshold` need not be provided. This is the currently recommended option for energy differences and valence and unoccupied eigenvalues.
- `zora` The ZORA approximation is used throughout the s.c.f. cycle, followed by a "scaled ZORA" [231] 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! ZORA (unscaled) is not the same as "atomic ZORA" and cannot be trusted. We also no longer recommend scaled ZORA values since there is no clear advantage. Just use `atomic_zora` unless there is need to do otherwise.

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

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

We recommend to simply use `atomic_zora` for all calculations, unless there is a particular need to stay with `relativistic none`.

If you really do want to use scaled ZORA (the case of `zora` keyword), 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.

Future version of FHI-aims will simply employ `atomic_zora` as the default level of relativity.

Tag: `use_spin_texture` (control.in)

Usage: `use_spin_texture Estart Eend`

Purpose: Calculate the spin texture of the bands fall in the designated energy range defined `Estart` and `Eend`.

`Estart` is the lower bound of the energy range of bands for which the spin texture will be calculated. `Estart` is defined with reference to the Fermi level (electronic chemical potential) calculated internally by FHI-aims. Unit: eV
`Eend` is the upper bound of the energy range of bands for which the spin texture will be calculated. `Eend` is defined with reference to the Fermi level (electronic chemical potential) calculated internally by FHI-aims. Unit: eV

This flag is presently based on the perturbative spin-orbit coupling method. Therefore, you should in the same time set:

`relativistic` atomic_zora scalar

`include_spin_orbit`

Spin texture is defined as the expectation value of the vector of Pauli matrices. The complete formulae we use, with examples, are given in the methods section and supporting information of Ref. [119]. By setting this flag, the code will calculate the expectation values for three components (σ_x, σ_y, and σ_z), which can help understand the spin polarization behaviour of individual bands. The values are printed out in separate files named `spin_texture.out` and `spin_texture.dat` which contain the same kind of data but simply in different formats.

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 an 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:

$$\sum_j h_{ij} c_{jl} = \epsilon_l \sum_j s_{ij} c_{jl} \quad . \tag{3.27}$$

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

FHI-aims now uses the open-source ELSI infrastructure http://elsi-interchange.org – and most often the efficient, massively parallel ELPA eigensolver (http://elpa.mpcdf.mpg.de) – to handle all aspects of this problem.

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 (publicly available as the ELPA library since 2011), 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 ≈1000 atoms (light elements) or ≈500 atoms (heavy elements, e.g. Au) in our experience. For large systems, there are alternative methods available through the ELSI library, including the orbital minimization method (libOMM), the pole expansion and selected inversion method (PEXSI), the shift-and-invert parallel spectral transformation eigensolver (SLEPc-SIPs), and the density matrix purification algorithms using sparse matrix linear algebra from the NTPoly library. Note that PEXSI and SLEPc-SIPs are not installed with FHI-aims by default.

The present section describes the available eigensolvers and density matrix solvers and relevant options in FHI-aims, including the determination of a Fermi level and occupation numbers for all orbitals following the process. Keywords starting with a prefix `elsi_` are ELSI-specific. The key ideas of using ELSI and its supported solvers are briefly introduced here. For more information, please refer to the ELSI documentation available at http://elsi-interchange.org.

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. (3.27) 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 overcon-

verged 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_{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. (3.27). For very large basis sets, this can lead to accidental ill-conditioning (some basis functions may be exactly expressable 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) or use the calculate_all_eigenstates keyword.

Tag: `calculate_all_eigenstates` (control.in)

Usage: `calculate_all_eigenstates`

Purpose: Specifies that all possible eigenstates obtainable from the basis set used (after ill-conditioning has been accounted for) should be calculated and stored.

This keyword instructs FHI-aims to calculate and store all possible eigenstates obtainable from the solution of the Kohn-Sham eigenvalue problem. It functions identically to setting the empty_states value to a large number, but makes the input file prettier.

Only users that know exactly what they're doing should use this option alongside an ill-conditioned basis set. Strange output may occur in this case, which may either be spurious or a symptom of a deeper problem. (This is true of ill-conditioning in general; this keyword only exposes it more openly.)

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. (3.27).

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

Important change : The naming scheme of the supported options has changed. Notably, the `lapack` and `scalapack` keywords are superseded by `serial` and `parallel`, respectively. The reason is that `lapack` and `scalapack` have indicated other linear algebra than custom (Sca)LAPACK in FHI-aims for a long time. For example, the `scalapack` option actually employed the ELPA eigenvalue solver, currently called through the ELSI infrastructure. The new naming scheme indicates linear algebra default options that may develop further over time, but these default options will not necessarily be tied to one and the same library going forward.

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

- `serial` : **Default serial eigensolver implementation**. Currently the serial eigensolver in ELSI, based on LAPACK and ELPA, will be employed.
- `lapack_fast` : Synonymous with `serial`.
- `lapack_2010` : LAPACK-based, and similar to the divide&conquer based standard solver provided by LAPACK itself.
- `lapack_old` : Expert solver provided by standard LAPACK. This is stable and not

a bottleneck in most standard situations.

- `lapack` : Disabled now.
- `parallel` : **Default parallel eigensolver implementation**. Currently the ELPA eigensolver will be called through the ELSI interface.
- `elsi` : Synonymous with `parallel`.
- `elpa` : Synonymous with `parallel`.
- `elpa_2013` : Same functionality as `scalapack_old`, however, substantially rewritten for an overall speedup and much improved scalability. 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_fast` : Synonymous with `parallel`.
- `scalapack_old` : Fully memory-parallel implementation of the eigenvalue solver based on ScaLAPACK itself, scales much worse than our own `scalapack_fast`.
- `scalapack` : Disabled now.
- `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 `parallel` 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" solver through the ELSI interface, which uses some ScaLAPACK infrastructure but has been rewritten from the ground up for much improved parallel scalability.

Note that a (separate) parallelization over k-points will be performed in periodic systems in any case.

`KS_method parallel` 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 and is the default where possible. For details, see keyword `collect_eigenvectors` .

Prior to the solution of Eq. (3.27) using the `serial` or `parallel` solvers, 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 sometimes 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: `elsi_method` (control.in)

Usage: `elsi_method method`

Purpose: Determines the usage of eigensolvers or density matrix solvers in ELSI. Must be compatible with the `density_update_method` keyword (see Sec. 3.6).

`method` is a keyword (string). Default: `ev`.

Available options for `method` are:

- `ev` : Use eigensolvers to solve the wave functions explicitly through ELSI. Supported serial solver is LAPACK. Supported parallel solvers are ELPA and SLEPc-SIPs (if compiled in). Compatible with all options of `density_update_method`.
- `dm` : Use density matrix solvers to directly compute the density matrix without explicitly solving the eigenproblem in Eq. (3.27). Note that this will not work with any post-processing that requires the wave functions. Supported solvers are ELPA, libOMM, PEXSI, SLEPc-SIPs, and NTPoly. Only compatible with `density_update_method density_matrix`.

Tag: `elsi_solver` (control.in)

Usage: `elsi_solver solver`

Purpose: Specifies the eigensolver or density matrix solver to use.

`solver` is a keyword (string). Default: `elpa`.

Available options for `solver` are:

- `elpa` : Direct, dense eigensolver ELPA (EigensoLvers for Petaflop Applications). Scales as $O(N^3)$ with respect to system size. Fast for systems of small and medium sizes (up to hundreds of atoms).
- `omm` : Density matrix solver libOMM (the Orbital Minimization Method). Scales as $O(N^3)$. No support for metallic systems. Not recommended for now.
- `pexsi` : Density matrix solver PEXSI (the Pole EXpansion and Selected Inversion method). Scales as $O(N^2)$ for 3D systems, $O(N^{1.5})$ for 2D systems, and $O(N)$

for 1D systems. Fast when solving a large system with sufficiently many MPI tasks (a thousand or more). PEXSI is not compiled with FHI-aims by default. To use it, either enable the compilation of PEXSI when building FHI-aims wiht CMake, or link FHI-aims against a precompiled ELSI library with PEXSI support.

- `eigenexa` : *Experimental* Direct, dense eigensolver EigenExa. The pentadiagonalization eigensolver eigen_sx in EigenExa can be faster than ELPA when solving the full eigenspectrum. Requires an externally compiled EigenExa library. No support for complex-valued problems. Therefore, in periodic calculations the number of k-points in any direction cannot be greater than 2.
- `sips` : *Experimental* Sparse eigensolver SLEPc-SIPs (the Shift-and-Invert Parallel spectral transformation method). Requires externally compiled SLEPc and PETSc libraries. Not recommended for now. No support for complex-valued problems. Therefore, in periodic calculations the number of k-points in any direction cannot be greater than 2.
- `ntpoly` : Density matrix purification algorithms implemented in the NTPoly library. For sufficiently large systems, scales as $O(N)$. Only competitive for thousands of atoms.
- `magma` : *Experimental* GPU-accelerated direct, dense eigensolvers in MAGMA. Drop-in enhancement to the eigensolvers in LAPACK. Requires an externally compiled MAGMA library.

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

Usage: `elsi_elpa_solver solver`

Purpose: Specifies the eigensolver used in ELPA.

`method` is an integer. Default: 2.

Available options for `solver` are:

- 1 : One-stage tridiagonalization eigensolver.
- 2 : Two-stage tridiagonalization eigensolver.

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

Usage: `elsi_elpa_n_single n_single`

Purpose: Specifies the number of s.c.f. steps in which the eigenproblems Eq. (3.27) are solved using single precision ELPA solvers.

`n_single` is an integer. Default: 0.

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

Usage: elsi_elpa_gpu gpu

Purpose: Switches on the usage of GPU (CUDA) acceleration in ELPA.

gpu is an integer. Default: 0.

Tag: elsi_omm_n_elpa (control.in)

Usage: elsi_omm_n_elpa n_elpa

Purpose: When using libOMM, specifies the number of s.c.f. steps in which the eigenproblems Eq. (3.27) are solved explicitly using ELPA. As an iterative solver, libOMM's performance heavily depends on the quality of the initial guess. By default, random numbers are used as inital guess. The eigensolution computed by ELPA proves to be a better choice.

n_elpa is an integer. Default: 6.

Tag: elsi_omm_flavor (control.in)

Usage: elsi_omm_flavor flavor

Purpose: Specifies the flavor of libOMM to be used.

flavor is an integer. Default: 0.

Available options for flavor are:

- 0 : Directly minimizes the OMM energy functional without transforming the generalized eigenproblem to the standard form before minimization. This is usually faster than flaver 2 if using several ELPA steps before switching to libOMM.

- 2 : Before OMM minimization, first transforms the generalized eigenproblem to the standard form using the Cholesky decomposition of the overlap matrix.

Tag: elsi_omm_tol (control.in)

Usage: elsi_omm_tol tolerance

Purpose: Specifies the convergence criterion of the OMM energy functional minimization.

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

Tag: elsi_pexsi_np_symbo (control.in)

Usage: elsi_pexsi_np_symbo np_symbo

Purpose: Specifies the number of MPI tasks assigned for the symbolic factorization step in PEXSI.

np_symbo is a positive integer. Default: 1.

Parallel symbolic factorization with more than 1 MPI task is not always stable, hence the default. Increasing np_symbo might accelerate the symbolic factorization, however might also cause a segfault. Note that the symbolic factorization step needs to be performed only once per s.c.f. cycle. Unless facing a memory bottleneck, using the default value is recommended.

Tag: elsi_eigenexa_method (control.in)

Usage: elsi_eigenexa_method method

Purpose: Specifies the eigensolver used in EigenExa.

method is an integer. Default: 2.

Available options for solver are:

- 1 : One-stage tridiagonalization eigensolver.
- 2 : One-stage pentadiagonalization eigensolver.

Tag: elsi_sips_n_slice (control.in)

Usage: elsi_sips_n_slice n_slice

Purpose: Specifies the number of slices used in SLEPc-SIPs. Note that the total number of MPI tasks must be a multiple of the number of slices. In practice, setting n_slice to be equal to the number of nodes seems to work well. The default value should always work, but by no means leads to the best performance.

type is a positive integer. Default: 1.

Tag: elsi_sips_n_elpa (control.in)

Usage: elsi_sips_n_elpa n_elpa

Purpose: Specifies the number of s.c.f. steps to be solved with ELPA. The performance of SIPs relies on a decent knowledge on the eigenvalue distribution, which is key to an efficient spectrum slicing. This can be calculated by ELPA in the first n_elpa s.c.f. steps.

type is an integer. Default: 0.

Tag: elsi_ntpoly_method (control.in)

Usage: `elsi_ntpoly_method method`

Purpose: Specifies the purification algorithm used in NTPoly.

`method` is an integer. Default: 2.

Available options for `method` are:

- 0 : Canonical purification.
- 1 : Trace-correcting purification.
- 2 : 4th order trace-resetting purification.
- 3 : Generalized canonical purification.

Tag: `elsi_ntpoly_tol` (control.in)

Usage: `elsi_ntpoly_tol tolerance`

Purpose: Specifies the convergence criterion of the density matrix purification.

`tolerance` is a small positive real number. Default: 10^{-4}.

Tag: `elsi_ntpoly_filter` (control.in)

Usage: `elsi_ntpoly_filter threshold`

Purpose: Specifies the threshold smaller than which the matrix elements will be discarded in the process of density matrix purification.

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

Tag: `elsi_magma_solver` (control.in)

Usage: `elsi_magma_solver solver`

Purpose: Specifies the eigensolver used in MAGMA.

`method` is an integer. Default: 1.

Available options for `solver` are:

- 1 : One-stage tridiagonalization eigensolver.
- 2 : Two-stage tridiagonalization eigensolver.

Tag: `frozen_core_scf` (control.in)

Usage: `frozen_core_scf boolean`

Purpose: Enables the frozen core approximation to reduce the dimension of the Kohn-Sham eigenproblem. Atomic basis functions whose eigenvalue is lower than `frozen_core_scf_cutoff` will be treated as core states. Useful for systems consisting of heavy elements. This keyword applies only to the solution of the Kohn-Sham eigenproblem. It does not imply a frozen core treatment anywhere else. See also `frozen_core` and `frozen_core_postscf`, which control the use of frozen core in other parts of the code.

`boolean` is either `.true.` or `.false.`. Default: `.false.`

Tag: `frozen_core_scf_cutoff` (control.in)

Usage: `frozen_core_scf_cutoff cutoff`

Purpose: Determines the number of core states when the frozen core approximation is enabled by `frozen_core_scf`. Atomic basis functions whose eigenvalue is lower than `cutoff` (eV) will be treated as core states.

`cutoff` is a negative number. Default: `-13605.5` (eV, which is about -500 Ha)

Tag: `frozen_core_scf_core_correction` (control.in)

Usage: `frozen_core_scf_core_correction boolean`

Purpose: Provides better accuracy for the frozen core states when the frozen core approximation is enabled by `frozen_core_scf`.

`boolean` is either `.true.` or `.false.`. Default: `.true.`

Tag: `frozen_core_scf_valence_correction` (control.in)

Usage: `frozen_core_scf_valence_correction boolean`

Purpose: Provides better accuracy for the unfrozen valence states when the frozen core approximation is enabled by `frozen_core_scf`.

`boolean` is either `.true.` or `.false.`. Default: `.true.`

Tag: `lopcg_adaptive_tolerance` (control.in)

Usage: `lopcg_adaptive_tolerance flag`

Purpose: *Experimental!* Allows the lopcg-algorithm to dynamically adjusts its convergence tolerance as $\max\{0.01\,|\delta n|,$ `lopcg_tolerance` $\}$ where δn 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: `mu_determination_method` (control.in)

Usage: `mu_determination_method type`

Purpose: Specifies the algorithm used to search for the Fermi level.

`type` is a descriptor (string) which specifies the desired algorithm to determine the Fermi level. Default: `bisection`

Available options are:

- `bisection` : Standard bisection algorithm. Usually robust to reach an accuracy of 10^{-13} in terms of electron count. If a desired accuracy cannot be reached by the bisection iteration, e.g., due to the limit of the machine precision, the remaining error (very small) will be arbitrarily cancelled out. Not compatible with the `integer` `occupation_type` .
- `zeroin` : Standard Brent's method. Not compatible with the `cubic` or the `cold` `occupation_type` .

Tag: `max_zeroin` (control.in)

Usage: `max_zeroin 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 Femi 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^{-13}.

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). 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 ϵ_l of each s.c.f. iteration, the selected occupation_type determines the Fermi level ϵ_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. [28] or other standard literature [137]. We only briefly list the available options for the occupation type here:

- gaussian : Gaussian broadening function [72]

$$f_l = 0.5 \cdot [1 - \mathrm{erf}(\frac{\epsilon_l - \epsilon_F}{\mathtt{width}})]$$

- methfessel-paxton : Generalized Gaussian-type distribution functions of Methfessel and Paxton (see Ref. [167] for details). In practice, any order beyond 1 is not recommended, and is not supported if bisection is chosen for the keyword mu_determination_method .

- fermi : Formally correct finite-temperature broadening scheme [166]

$$f_l = \frac{1}{1 + \exp[(\epsilon_l - \epsilon_F)/\mathtt{width}]}$$

  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.

- integer : Forces the occupation numbers to be integers.

- `cubic` : *Experimental* Cubic polynomial broadening.
- `cold` : Cold smearing technique proposed by Marzari and Vanderbilt.

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 [137, 77, 235]. 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. This is a non-linear optimization problem and not always trivial.

The most important keywords related to this problem in FHI-aims are `adjust_scf`, which is set by default and automates the process with a choice of s.c.f. settings that is often safe. The key parameters that can be manually adjusted are `charge_mix_param` and `occupation_type`. Many more keywords are described below, but usually, these are the relevant choices.

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.

It is very important to remember that different stationary densities for the exact same atomic geometry and for the exact same density functional are a real and not always unrealistic possibility in DFT. A simple example are antiferromagnetic vs. ferromagnetic spin states in some systems. In such cases, the true ground state in a DFT sense is the stationary density that yields the lowest energy. It can be found by way of a global search for different stationary densities, usually by varying the initial density guess.

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. [28] 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 `initial_moment` of 2 (far lower than the Hund's rule value) for each Fe atom 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 stop if no initial moments of any kind are provided by the user. Setting at least one individual `initial_moment` tag in `geometry.in`, or both will allow you to run.

- *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. *However, for very large systems the Kerker preconditioner can cost significant amounts of time – see the detailed timing output that is written by the code at the end of each s.c.f. iteration. You may try to switch it off.* 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 bohr^{-1}).

- The keyword `sc_init_iter` sets the number of s.c.f. iterations after which the Pulay mixer resets itself from scratch. This can significantly help in cases of bad convergence. If you have real mixer trouble, please consider this keyword.

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.

For further suggestions to improve s.c.f. convergence, see Sec. A.8.

3.10.1 Visualizing the convergence of the s.c.f. cycle

There is a simple tool that can be used to visualize the s.c.f. convergence behavior of FHI-aims graphically for a given run. Preferably do this analysis on your desktop computer (i.e., copy over the necessary files). This is what you need to do:

- Install the Grace 2D visualization program (http://plasma-gate.weizmann.ac.il/Grace/) on your computer.

- Go to the directory with the FHI-aims output file you wish to analyze.

- From the FHI-aims *utilities* directory, copy over the file `scf_convergence_template.agr` .
- At the commandline, call the FHI-aims utility `plot_scf_convergence.pl` [FHI-aims output file] to visualize the s.c.f. convergence behavior of the FHI-aims output file. `plot_scf_convergence` can be found in the FHI-aims *utilities* directory and must (of course) be called with the correct directory path preceding the file name.

If successful, this procedure will assemble and open a graphical representation of the s.c.f convergence of FHI-aims.

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: Zero, 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, ...).

Tags for general section of `control.in`:

Tag: `adjust_scf` (control.in)

Usage: `adjust_scf frequency number`

Purpose: Adjusts key parameters that govern the s.c.f. cycle based on a rough estimate of the system's band gap.

`frequency` is a keyword, `once`, `never`, or `always`. Default: `once`.

`number` is an integer number (zero or greater). Default: 2.

This keyword decides whether key s.c.f. convergence parameters will be adjusted automatically during the s.c.f. cycle, based on a simple estimate of the system character according to its approximate HOMO-LUMO gap or band gap.

The `number` keyword determines in which iteration of the s.c.f. cycle the adjustment will be attempted. A zero value corresponds to the initial s.c.f. cycle; values of 1, 2, etc. correspond to an update after the first, second, etc. s.c.f. iterations are almost complete and their eigenvalue spectra known.

The `frequency` keyword determines for which full s.c.f. cycle (i.e., for which geometry step) an adjustment will be made:

- `once` means that an adjustment of the s.c.f. parameters will only be made in the first geometry in a geometry relaxation or MD run.
- `always` indicates that an adjustment will be made for every new s.c.f. cycle, i.e., for every new geometry in a run.
- `never` indicates that no adjustment will be attempted.

Parameters will only be adjusted if they are not explicitly set by a keyword in `control.in`. Any parameter that is included in `control.in` will not be modified by the `adjust_scf` keyword. For `frequency once` or `always`, the following parameters may be adjusted:

- The initial default value of `charge_mix_param` for `mixer pulay` will be set to 0.05 (i.e., an overall cautious value). For `frequency never`, the default value of the `charge_mix_param` keyword remains at its usual default value 0.2 (for many metallic or spin-polarized systems, this is a fairly aggressive value).
- In s.c.f. iteration `number`, the current estimated value of the HOMO-LUMO gap (for solids, the band gap) is checked. If the system shows fractional occupation numbers or if the estimated gap has a value of less than 0.2 eV at this point, the system is likely near-degenerate or metallic and the s.c.f. cycle could be diffcult to converge. In this case, the `charge_mix_param` is set to 0.02 – a cautious value but, in conjunction with the Pulay `mixer`, still surprisingly effective. The broadening of the occupation numbers near the Fermi level is increased to `occupation_type option` 0.05 [eV].

- If, instead, the gap is found to be equal or greater than 0.2 eV in s.c.f. iteration number, the rather aggressive default charge_mix_param 0.2 is kept for the Pulay mixer, and the default broadening value (suitable for non-metallic systems) occupation_type option 0.05 [eV] is also kept.

Tag: allow_restart_xc_pre (control.in)

Usage: allow_restart_xc_pre flag

Purpose: This keyword will safeguard against xc_pre being accidentally set together with elsi_restart (they can still be set together on purpose if this keyword is .true.)

flag is a logical variable (.true. or .false.). Default: .false.

The keywords xc_pre and elsi_restart both allow one to specify a non-standard (and usually faster) initialization of the electronic structure that is later calculated self-consistently for the exchange-correlation method specified by the usual xc keyword.

One frequent problem is that they can be accidentally set togethe. In that case, the initialization using xc_pre can undo the often highly valuable initialization already present from the elsi_restart information. For example, xc_pre can replace the highly expensive information from a previous hybrid DFT run and replace it with the much less suited information from, say, a PBE pre-initialization.

This is, ostensibly, a user error, but can cost immense amounts of CPU time especially for the highest-value calculations. Therefore, we safeguard against this possibility.

When the xc_pre keyword is set, elsi_restart can still be used at the same time – but only if no elsi_restart restart files are actually available to read (i.e., if all that elsi_restart would do is write). If elsi_restart actually does have information to initialize the calculation, keyword xc_pre will only be allowed to be used if allow_restart_xc_pre is explicitly set to be true. If it isn't, then the code will stop and explain to the user that the two should not be used together.

The allow_restart_xc_pre keyword has no effect when used together with the older restart keyword.

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. Now set by the adjust_scf keyword.

See Ref. [28] 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 μ, $n_{\mathrm{KS}}^{(\mu)}$, with the (already

mixed) *input* density that defined those equations, $n^{(\mu-1)}$:

$$n^{(\mu)}_{\text{dmp}} = n^{(\mu-1)} + \hat{G}^1(n^{(\mu)}_{\text{KS}} - n^{(\mu-1)})\,. \tag{3.28}$$

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 general, please also see Appendix A.8, since several keywords can be used to alleviate s.c.f. mixing instabilities.

- 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 adjusted according to the adjust_scf keyword, depending on the estimated band gap / HOMO-LUMO gap. For non-metallic systems, we choose a conservative value of 0.2. In metallic systems or systems that are otherwise problematic, the default value set by adjust_scf is value=0.02. Note that this small value does not necessarily correspond to slow mixing since the Pulay mixer will learn over time and accelerate the mixing process.

- 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 still important and may be chosen around 0.05.

See also the mixer and preconditioner keywords.

Tag: `relative_fp_charge_mix` (control.in)

Usage: relative_fp_charge_mix value

Purpose: Parameter for under-relaxation of the fixed point part of s.c.f. cycle with the broyden mixer

value is a real number between 0. and 1. Default: 0.05

relative_fp_charge_mix determines the under-relaxation of the fixed point part of the Broyden mixer s.c.f. cycle together with charge_mix_param. relative_fp_charge_mix and charge_mix_param are multiplicative, and if no history is included the effective under-relaxation is relative_fp_charge_mix times charge_mix_param.

Tag: `default_initial_moment` (control.in)

Usage: default_initial_moment moment

Purpose: For spin-polarized calculations, sets a per-atom default initial moment fo each atom that makes up the initial electron density.

moment is either a string or a number that defines the desired initial difference of the number of spin-up vs. spin-down electrons on each individual atom, $N^{\uparrow} - N^{\downarrow}$. Default: hund for isolated atoms. Zero otherwise.

Sets the default initial spin moment per atom for all atoms whose initial_moment s are not specified explicitly in geometry.in.

However, do not use the default_initial_moment keyword unless you know exactly what you are doing. default_initial_moment is a very dangerous keyword if used unwisely. It sets a per-atom moment. Remember that, if a structure contains (say) ten atoms, the initial moment per structure will be ten times the default_initial_moment per atom - possibly, assigning entirely unphysical initial spin moments to atoms that should never have seen a local spin polarization in the first place. The result will be a terrible s.c.f initialization, leading to unphysical results or to no s.c.f. convergence at all.

Instead, think about which exact atoms should carry an initial spin moment in your structure, and what that moment should be. Then, use initial_moment keywords for each atom in your geometry.in file. Doing so will usually allow you to specify a much better initial spin density (and, thus, better s.c.f. convergence) than any blanket default_initial_moment keyword can ever do.

We keep the default_initial_moment keyword for convenience when dealing with some magnetic materials. However, with a single, unfortunate choice of default_initial_moment , one can initialize a complex structure with a completely unphysical spin density. This will lead to severe issues with the s.c.f. cycle.

If there is at least a single initial_moment keyword specified in geometry.in and if the keyword default_initial_moment is not set explicitly, then the default_initial_moment will be zero for all atoms for which no initial_moment is specified explicitly.

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 entire 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.

Only if no initial_moment is included in geometry.in at all in a spin -polarized calculation, the default_initial_moment must be specified explicitly by the user for the code to run at all.

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. However, it is much better (and often the only way) to use the force_occupation_basis keyword to specify the expected orbital occupation of a free atom explicitly. Many free atom calculations will not converge in a s.c.f. cycle if the expected symmetry breaking (and, thus, orbital occupation) is not specified.

Warning: It is *not* advisable to set a default_initial_moment other than zero

in structures in which only a few atoms actually carry spin (such as a large molecule with a few transition metal atoms). Rather, a good choice would be to set a zero `default_initial_moment` and then set finite `initial_moment` values for individual atoms in `geometry.in`.

And again: Setting `default_initial_moment` to zero and *not* specifying any `initial_moment` values in `geometry.in` will lead to a zero spin state in the final result, simply because the system is never given any indication which way to break its symmetry. So one does need to set a finite moment somewhere if a finite-spin converged solution is expected in the calculation.

The upshot is: It pays to think about the right spin initialization. Do not use the `default_initial_moment` keyword unless you know exactly what you are doing. Instead, use individual `initial_moment` keywords in `geometry.in`. There may be multiple different self-consistent solutions, and in spin-polarized systems, this can happen for very natural reasons (e.g., ferromagnetic vs. antiferromagnetic states). Similar to geometry optimization, starting from a very bad initial guess can cause terrible problems, particularly for s.c.f. convergence; but physically unreasonable solutions may also result. Conversely, a good starting point may greatly simplify a calculation. It really does pay to think about a detailed, physically reasonable spin initialization.

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-self-consistent solution for a superposition-of-atoms potential that yields only the sum of eigenvalues as a result. If a non-self-consistent 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-self-consistent`: 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`, `pulay`, or `broyden`. Default: `pulay`.

FHI-aims provides three mainstream density mixing algorithms across the s.c.f. cycle, `type linear`, `pulay`, and `broyden`. We here only give a brief summary of options,

please see Ref. [28] for further references and for the exact mathematical details.

For most practical purposes (non-pathological systems), Pulay's DIIS mixing algorithm [193] 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,...,n) to be mixed with the Kohn-Sham output density of iteration μ. `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 `fixed_spin_moment` 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. Here, demanding a specific orbital occupation using the `force_occupation_basis` keyword 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.

The `broyden` `mixer` is an improvement on the `linear` `mixer` . The `broyden` `mixer` works by effectively dividing the s.c.f. cycle into two separte parts: the space in which we have local information gained by previous evaluations of the s.c.f. cycle,

and the remaining space in which we do not have information. charge_mix_param determines the linear factor which under-relaxes the density predicted by the broyden mixer , n_max_broyden *n* controls the number of past iterations used to construct the next estimate, and relative_fp_charge_mix is the additional (multiplicative) under-relaxation affecting only the step length in the space where we have no further information.

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: `n_max_broyden` (control.in)

Usage: `n_max_broyden number`

Purpose: The number of past iterations that the broyden 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 Broyden matrix, which can become near-singular.

Note that the storage effort associated with Broyden 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_broyden 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 bohr^{-1}).

See Ref. [28] regarding the mathematical definition of the Kerker-type [159, 174, 128] 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^{-1}). Typical values in the literature range around 1.0-2.0 bohr^{-1}, 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 $n_{\mathrm{KS}}^{(\mu)}$ (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.

- dielectric : The inverse of the microscopic dielectric function, $\epsilon^{-1}(\mathbf{r}, \mathbf{r}'; 0)$ is used to precondition the density. This option is experimental and computationally expensive but the the dielectric function can be theoretically justified to be a good preconditioner. Works only for non-periodic systems.
- none or off : No preconditioner used.

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. [28] 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: kerker_factor (control.in)

Usage: kerker_factor value

Purpose: value is a real positive number. Additional empirical factor for the last step of the kerker precondioner. Best results were achieved by choosing value as the characteristic momentum $(q_0 - 0.5)$ associated with the Thomas-Fermi type screening assumed in the kerker preconditioner (kerker preconditioner).

This keyword is experimental and derived fully empirically. For notorious cases the keyword may help to achieve convergence (faster). For $q_0 = 1$ the method is identical to the standard kerker preconditioner.

Tag: restart (control.in)

Usage: restart file

Purpose: Saves and reads the final wave function or density matrix of each scf-cycle to/from file.

file is a string, corresponding to the desired restart filename.

Note: A more generic restart functionality is provided by the elsi_restart – please see that keyword for a description. The elsi_restart keyword is the supported restart functionality going forward unless you have special reasons to prefer the simple restart keyword.

The restart keyword summarizes a set of different cases for which the electronic structure information in FHI-aims can be saved and then later used to restart a calculation from that point. restart comes in many internal variants and, for example, restarting a calculation with more MPI tasks than a previous run is not possible. The alternative elsi_restart keyword (which uses a different storage format than the restart keyword) is much more flexible and may be the better choice.

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. Mind that restart is currently not supported for keywords load_balancing and use_local_index so that file will not be written or read when either keyword is set.

It is important to note that the restart infrastructure corresponds to a restart from the last Kohn-Sham orbitals, not from the last density. In practice, this means that the code will restart from the last *unmixed* Kohn-Sham density, not from the last *mixed* density. When restarting from a non-self-consistent starting point, this can lead to unexpected jumps in the calculated non-self-consistent total energy. Only the self-consistent total energy is truly meaningful and this (the self-consistent) total energy should be the same for the same stationary density. (Note also that some systems may exhibit several different self-consistent stationary densities – a simple example are antiferromagnetic vs. ferromagnetic spin states in some systems. In such cases, the true ground state in a DFT sense is the one with the lowest energy and must be found by varying the initial density guess.)

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 MD_restart for more information on the separate restarting process for molecular dynamics.

A much more complete overview of the restart infrastructure in FHI-aims can be found in the dedicated Section 4.7.

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).

Note: A more generic restart functionality is provided by the elsi_restart, described further below. We recommend to use elsi_restart unless you have special reasons to prefer the simple restart keyword.

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.

Note: A more generic restart functionality is provided by the elsi_restart, described further below. We recommend to use elsi_restart unless you have special reasons to prefer the simple restart keyword.

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: `force_single_restartfile` (control.in)

Usage: force_single_restartfile .true.

Purpose: Forces FHI-aims to always write a single, wavefunction based restart file if possible.

Only relevant when using keyword restart. For technical and efficiency reasons FHI-aims uses different types of restartfiles depending on the eigenproblem solver (see Section 4.7 for details). In cases where the wavefunction is needed (e.g. for external post-processing, ...), the default restart handling might make it difficult to do so.

This option only works for cluster (non-periodic) and periodic Γ-only calculations.

Tag: `elsi_restart` (control.in)

Usage: elsi_restart task freq

Purpose: Controls the density-matrix-based restart using parallel matrix I/O routines provided by the ELSI software. Although they are not known to interfere with each other, the two restart keywords, restart and elsi_restart, should not be requested in the same calculation.

task is a string, specifying the desired restart task.

freq is a positive integer, specifying the frequency of outputting restart info.

Available options for task are:

- write : Writes restart info (in ELSI format) to files. The info will be written to files in every freq s.c.f. iterations, and will always be written out after a converged cycle. The number of files depends on the choice of elsi_restart_use_overlap .

- read : Reads restart info from files and uses it (instead of free atom superpositions) as the initial guess of an s.c.f. cycle. The code will search for files written by the write option. If relevant files do not exist, the code will abort. freq is not used for this option. The restarted run can use any number of MPI tasks, i.e., not necessarily the same number as used in the original run.

- read_and_write : Performs what the write and read options do. Note that if the restart files do not exist, the code will still proceed normally.

Restarting hybrid functional calculations from the density matrix obtained using a different (i.e. less expensive) functional may reduce the number of SCF steps taken by the hybrid calculation. However, hybrid functional calculations currently require that the symmetry_reduced_k_grid be set to .false.. If you are planning to restart a calculation with a hybrid functional from a less-expensive functional, first use elsi_restart combined with symmetry_reduced_k_grid .false. for initial SCF convergence with the less expensive functional, then restart your calculation with the hybrid functional. Note also that if a calculation is restarted using a different functional than that used to obtain the density matrix, the atomic_solver should be set to that used to obtain the density matrix.

Note that the elsi_restart keyword – when it has actual past restart information available to read – cannot be used together with the xc_pre keyword, unless explicitly allowed using the allow_restart_xc_pre (see there for more information). We do this to prevent unintended duelling inializations. The two keywords can still be used together if all that elsi_restart would do is write information, not read it.

Tag: elsi_restart_use_overlap (control.in)

Usage: elsi_restart_use_overlap boolean

Purpose: By default, elsi_restart uses density matrices as its restart info. The number of density matrix files is equal to the number of k-points times the number of spin channels. The density matrix files can be used to restart a calculation of the same geometry. elsi_restart_use_overlap should be used to initialize a calculation with the density matrices obtained from a different structure. When this keyword is set to .true., the code writes overlap matrix files in addition to density matrix files. The number of overlap files is equal to the number of k-points. The stored overlap matrices are used to extrapolate the stored density matrices.

boolean is either .true. or .false.. Default: .false..

Tag: ri_density_restart (control.in)

Usage: ri_density_restart task value

Purpose: Allows a calculation to be restarted from an RI decomposition of the density. The product (auxiliary) basis is used as the basis for this decomposition. This restart is not designed to be used to restart a calculation partway through an SCF loop, since the RI decomposition of the density is not exact. Instead, it can be used to start a calculation from a converged density while using (for example) a different functional or k-grid. Note that if a calculation is restarted using a different functional than that used to obtain the RI density, the atomic_solver should be set to that used to obtain the RI density. It is also used to produce training data for and read in electron densities predicted by the machine-learning code SALTED. task is a string, specifying the desired restart task.

value If task is write, value is an optional positive float which defines a cutoff radius for calculating the overlap of the product basis functions. Default: 1.5. If task is read, value is an optional non-negative integer specifying the maximum number of SCF steps to be performed after reading the density. Default: sc_iter_limit .

Available options for task are:

- write : Writes restart coefficients to a file 'ri_restart_coeffs.out'. The information will be only be written to files after a converged cycle. value defines a cutoff radius for each product basis function, such that basis functions centred on an atom further than this distance from the current integration point are neglected when calculating the overlap matrix. This radius is given by value multiplied by the charge radius of the product basis function in question. The default value should always be a safe choice.

- read : Reads restart info from file 'ri_restart_coeffs.out' and uses it (instead of free atom superpositions) to build an approximate density as the initial guess of an s.c.f. cycle. If relevant files do not exist, the code will abort. value sets

sc_iter_limit ; if set to 0, the calculation will be initialised with the RI density but no SCF steps will be completed; the density will be assumed to be converged.

- read_and_write : Performs what the write and read options do. Note that if the restart files do not exist, the code will still proceed normally.

Tag: **ri_full_output** (control.in)

Usage: ri_full_output boolean

Purpose: When this keyword is set to .true., the code writes overlap matrix and density projection files in addition to density matrix files, along with information about the product basis needed to interface with the SALTED machine-learning package, and the density and partition table on the internal real-space grid used by FHI-aims. This only applies when ri_density_restart is supplied with write.

boolean is either .true. or .false.. Default: .false..

Tag: **calc_dens_superpos** (control.in)

Usage: calc_dens_superpos .true.

Purpose: When reinitializing the SCF cycle, fall back to the superposition of free atoms density for the initial guess (instead of using the guess from the previous converged cycle).

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, `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. [28] for a brief discussion). A typical tight convergence criterion is `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 EITHER 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, OR a string, '`not_checked`'.

Default: `not_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.

One can explicitly set the keyword value to the string '`not_checked`'. In this case, the forces are computed in only a single shot, their s.c.f. convergence will not be checked.

In this case, the code now relies on the default convergence criterion for the electron density itself (`sc_accuracy_rho`) to be sufficiently tight to guarantee good forces. This avoids excessive numbers of force evaluations in production runs.

Calculating forces is relatively expensive, so this convergence criterion is either not checked (default behavior), or it is 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.

For simple structure relaxation, not_checked will often be good enough if the electronic criteria (especially sc_accuracy_rho) are set reasonably tightly.

For molecular dynamics, however, even slightly underconverged forces may lead to long-term energy drifts in (nominally) constant-energy runs. Here, it is a good idea to try out in explicit tests how tightly sc_accuracy_eev must be set to guarantee drift-free trajectories.

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.

Default: Between 1d-6 and 1d-3 e/$a_0{}^3$, depending on number of atoms or if a forces corrected calculation is taking place (see below).

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, $n_{\mathrm{KS}}^{(\mu)}$, is checked against the input density $n^{(\mu-1)}$ to the same equations. A typical tight convergence criterion is value=10^{-5}.

sc_accuracy_rho is the most important s.c.f. convergence criterion. If the keyword is not set in control.in, the code chooses its initial default as follows:

1. In general, but especially if the stress tensor is needed and/or if the output keyword is used, use fairly tight criteria:
   - Up to 6 atoms in geometry.in: value=10^{-6}
   - Between 6 and 6000 atoms in geometry.in: value=$10^{-6} \cdot \sqrt{\mathrm{n\_atoms}/6}$
   - Above 6000 atoms in geometry.in: value=$10^{-6}\sqrt{1000}$
2. If forces are corrected for s.c.f. convergence incompleteness by the force_correction keyword or if forces are not used at all:
   - Multiply the defaults of item 1 by a factor of eight, since the convergence of total energies or of corrected forces is already very good.

Tag: `sc_accuracy_stress` (control.in)

Usage: `sc_accuracy_stress value`

Purpose: Convergence criterion for analytical stress.

`value` is EITHER a small positive real number [in eV/Å^3], against which the maximum difference of the analytical stress components between the present and previous s.c.f. iteration is checked. A negative number, or simply the string 'not_checked' results in no convergence check.

Default: `not_checked` .

Warning. Setting `sc_accuracy_stress` to a finite non-negative value will result in a disproportionately large computational cost. In this case, the number of stress evaluations per relaxation step is at least doubled. This is normally by far the most expensive part of the calculation. Only set this to a finite value if you have a good reason to do so.

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 keyword 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_init_factor` (control.in)

Usage: sc_init_factor number

Purpose: The sc_init_iter keyword will not trigger if the density convergence criteria are already within a factor sc_init_factor of the density convergence criterion, sc_accuracy_rho .

number is an real (double precision) number. Default: 1.d0

See the sc_init_iter keyword for a more complete description of this behavior.

Tag: **sc_init_iter** (control.in)

Usage: sc_init_iter number

Purpose: If the s.c.f. cycles for the initial geometry of a run fails to converge within (number) iterations, then FHI-aims will end this s.c.f. cycle and begin a new one with the exact last wave function as its starting guess.

number is an integer number. Default: 1001

sc_init_iter ends only the s.c.f. cycle for the first geometry of a run. The idea is to do a step that looks exactly like a new geometry step, but to not alter the geometry. Rather, reinitializing from the exact orbitals reached after (number) iterations ensures that the mixer and other parts of the calculation will not drag along some misguided information of the original superposition-of-free-atoms initialization. In some cases, such a clean start can help converge the s.c.f. cycle of a calculation that otherwise has difficulties to converge at all.

See also the sc_init_factor keyword, which can be used to tune this behavior further.

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.

Tag: **apply_boys** (control.in)

Usage: apply_boys KSmin$_\alpha$ KSmax$_\alpha$ KSmin$_\beta$ KSmax$_\beta$ integer

Purpose: In the cluster case is used to switch on a subspace localization using the Boys localization algorithm.

KSmin$_\alpha$ and KSmax$_\alpha$ are the integer KS-state indexes which define the lower and upper boundary of the subspace which should be included in the transformation. In calculations without spin, the β parameters are omitted.
integer is an integer and can either be 0, 1, or 2. If set to 0, the localization is performed at the beginning of the SCF cycle. If set to 1, it is performed throughout the cycle and if set to 2, it is performed at the end (before writing the restart information).

This is an experimental keyword that allows to perform a Boys localization on one or more subspaces of the KS eigenvector. Boys localization is only applicable in non-periodic calculations. Since each localization requires calculation of the transition dipole matrix, this adds a considerable overhead in computation time if it is performed in each SCF cycle. The currently recommended setting is either 0 or 2.

Tag: xc_pre (control.in)

Usage: xc_pre xc-type steps

Purpose: Specifies the exchange-correlation approach used during the initial steps of a self-consistent DFT / Hartree-Fock.

xc-type is a keyword (string) which specifies the chosen exchange-correlation functional.
steps is an integer that determines the number of SCF steps performed with this functional before switching to the functional defined by xc .

One can specify any xc-type from the LDA, GGA and meta-GGA options available as listed for xc . One can also use the dfauto formalism, i.e. xc_pre dfauto xc-type steps for non-hybrid XC functionals. *See the dfauto section for tag xc for more detail.*

Note that the elsi_restart keyword, when it has actual past restart information available to read, cannot be used together with the xc_pre keyword, unless explicitly allowed using the allow_restart_xc_pre (see there for more information). We do this to prevent unintended duelling inializations. The two keywords can still be used together if all that elsi_restart would do is write information, not read it.

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 $\boldsymbol{R}_{\mathrm{at}}$. 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.

Before coming to the full set of related keywords, we give some basic comments below. Please also consider using (and adding to) the wiki, especially if you encounter any kind of behavior that is obviously not intended. We would like to know about such cases.

One "most important" rule:

For efficient local structure optimizations, do not simply use "tight" settings from an arbitrary starting geometry and wait.

It is usually *much* more effective to use a two-step approach. First, use "light" settings in a pre-relaxation step to get the rough geometry right quickly. Then, use "intermediate" or "tight" settings for post-relaxation or post-processing only, e.g., based on the "geometry.in.next_step" file that is written out by the code by default (see below).

Basic handling:

For most applications, reliable structure optimizations of atomic coordinates into a local minumum of the potential energy surface can be obtained by simply setting the keyword `relax_geometry bfgs threshold`
`threshold` denotes the minimum absolute force component in eV/Å acting on any atom. Typically, a `threshold` value of 5d-3, i.e., $5{\cdot}10^{-3}$ eV/Å, corresponds to a very tightly converged structure optimization. Do not use much lower values since you might spend substantial amounts of CPU time for no measurable gain.

For unit cell optimizations in periodic systems, the keyword `relax_unit_cell` must additionally be invoked (otherwise, the unit cell shape will remain fixed).

Finally, individual coordinates or degrees of freedom can be fixed by using the keyword `constrain_relaxation` .

Another method to run a constrained relaxation is the combined use of the keywords `symmetry_n_params` , `symmetry_params` , `symmetry_lv` and `symmetry_frac` . When this block is added to a `geometry.in` and `relax_geometry` (and optionally `relax_unit_cell`) are set in `control.in`, a geometry relaxation is started that is con-

strained to a parameter reduced space defined by the user. This can be used to enforce a symmetry-preserving relaxation by providing the exact prototype of the given structure in its analytic form. It furthermore allows for the introduction of local symmetries or local symmetry breaking like distortions. **Important:** When using these constraints *all* keywords must be specified for *all lattice and atomic degrees of freedom*. If you want to run with these constraints on only the lattice or the atomic positions, add the other block fixing the coordinates or with $3N_{Lattice}$ or $3N_{atomic}$ additional parameters for a free relaxation. **Important:** The relationship between the full coordinates and the reduced parameters has to be linear, i.e. there is a transformation matrix A and a transformation matrix B that can map the flattened 1-dimensional representation of the lattice cell/atomic positions to the parameter-reduced space. For the monoclinic and triclinic lattice systems, e.g. do not use $c \cdot \cos(\beta)$ and $c \cdot \sin(\beta)$ but replace them with independent variables e.g. d and e. *It is on the to-do-list to make aims recognize these cases internally.*
Hint: You can create `geometry.in`'s already containing the required input block for a symmetry-constrained relaxation using AFLOW as of version 3.1.204. It gives access to hundreds of structure prototypes in the AFLOW Prototype Library [164, 107] throughout all spacegroups. Simply add `--add_equations` to the command. Example:

```
aflow --proto=AB_cF8_225_a_b --params=5.64 --aims --add_equations
```

To ensure the constraints are mapped onto the positions of the atoms inside of the unit cell (AFLOW's default behavior) use the `aflow_structure_wrapper` utility in `utilities` Example:

```
aflow_structure_wrapper --proto=AB_cF8_225_a_b --params=5.64 -o geometry.in.aflow
```

To obtain gradients for a given structure, but without any subsequent structure optimization, molecular dynamics run, etc., employ the keywords `compute_forces`, `compute_analytical_stress` or `compute_numerical_stress` instead.

`relax_geometry` `bfgs value` calls a trust radius enhanced variant of BFGS, also available as `trm`. This variant covers all use cases.

The `bfgs` algorithm can be tweaked by specifying an initial guess for the Hessian matrix, which can influence the performance. The default is the model Hessian matrix by Lindh and coworkers,[155] which leads to reliable and fast optimizations in the vast majority of scenarios.

If, for any reason, the BFGS algorithm does not perform well for a given structure initially, a number of options are available. The standard first step should always be to simply restart the optimization from the updated geometry and Hessian matrix in `geometry.in.next_step` (see below). Alternatively, one may preset a simple, diagonal Hessian using the keyword `init_hess diag value` (see the actual keyword description for more details). Finally, the `energy_tolerance` and `harmonic_length_scale` keywords decide when the `bfgs` algorithm will abort a relaxation because a presumed deviation between the predicted and the real energy landscape. They can be set to more forgiving values if needed.

For the relax_geometry bfgs algorithm, the handling of *restarting* relaxations is formalized by writing out a file geometry.in.next_step after each relaxation step. By default, this file contains both the geometry and the current estimate of the Hessian matrix of the system that are used by the BFGS algorithm. For an organized restart, simply copy this file to geometry.in, and the stored geometry and Hessian will be used to reinitialize the BFGS algorithm if the control.in file requests a structure relaxation. If the keyword distributed_hessian is in use, the current estimate of the Hessian matrix will be stored in a separate binary file, which should not be modified by hand.

Use the 'get_relaxation_info.pl' utility script to monitor the progress of an ongoing or finished relaxation run. This information can be immensely helpful to make sure that you are not, e.g., spending your time optimizing the last 10^{-5} eV out of an already converged structure relaxation.

To stop an ongoing structure relaxation in an organized way, create the abort_opt file in the respective directory.

MP2 and RPA total energy derivatives with pz-lda and pbe potential:

For running particle-hole RPA total energy derivatives calculation following tags in control.in file are necessary: One should set (keyword rpa_force freq_formula_method or matrix_diag_method) depending on two different approaches. The first one use frequency integral formula for evaluation of correlation energy which scale N^4, with N being the system size. The later one scales N^6, comparatively consume more time as compared to first one. Others additional tags required are as RI_method lvl, use_2d_corr .false., DFPT vibration or vibration_reduce_memory if doing calculation with pz-lda, with pbe only use vibration_reduce_memory, total_energy_method rpa, this is optional does not effect anything. Calculation with bigger Gaussian basis set, the accuracy is affected with the choice of number of auxiliary basis sets control by tag prodbas_acc , it is highly recommended to use larger values like 1.E-2. Currently, MP2 and RPA forces are evaluated for closed shell systems only. For MP2 force, same (keyword rpa_force mp2_force) invoke. Frozen-core approximation can be invoked by using additional tag frozen_core_postSCF 1.

The RPA+SE and RPA+rSE total energy derivatives:

The forces at the RPA+SE (RPA+single excitation) and RPA+rSE (re-normalized single excitation) level can also be calculated by using additional tag to RPA calculations by setting post_SCF_force to SE or rSE. Additionally, for RPA, RPA+SE and RPA+rSE calculations for big molecules and with bigger basis set the tag force_reduce_memory .true. will be quite helpful.

The RPA single point energy with RI-LVL can be computed by using tag rpa_ene_lvl .true. . For this purpose total_energy_method rpa will only work.

Stress tensor:

For unit cell optimization (keyword relax_unit_cell), an analytically computed stress tensor will be used where available, thanks to work by Viktor Atalla, Christian

Carbogno, and Franz Knuth [133]. For some density functionals, the analytical stress tensor is not available. In these cases, the unit cell itself can still be relaxed, but the stress tensor must be computed numerically from finite differences of total energies.

The numerical finite-difference stress tensor is always available as a fallback.

Finally, we note that, in some cases, just optimizing the unit cell shape blindly may not be what you want. For example, in high-symmetry structures a fit to Murnaghan's Equation of state—see the utility provided in the *utilities* directory—will be more accurate and give you more information about the system (bulk moduli and, in case of more than one phase, also access to transition pressures).

Hybrid functionals:

For hybrid functionals, analytic energy gradients and the analytic stress tensor are only available together with the `RI_method LVL_fast` version of "resolution of identity" of the two-electron Coulomb operator.[116]

For perturbative methods (MP2 perturbation theory, RPA and beyond), analytical gradients are not yet available.

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 3×3 block. The option `block` consists of 9 numbers in column-first (Fortran) order. The 3×3 block corresponding to `j_atom`, `i_atom` is initialized by the transposed of `block`. The Hessian matrix is input in units of eV/Å^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 [155]. Please note that the Lindh model matrix is now also directly available with `init_hess Lindh`.

Tag: `hessian_block_lv` (geometry.in)

Usage / purpose: Like `hessian_block`, but for Hessian matrix elements between lattice vector degrees of freedom.

Tag: `hessian_block_lv_atom` (geometry.in)

Usage / purpose: Like `hessian_block`, but for Hessian matrix elements between lattice vector degrees of freedom.

Tag: `hessian_file` (geometry.in)

Usage: `hessian_file`

Purpose: In `geometry.in`, this keyword indicates that there exists a `hessian.aims` file to be used to construct the Hessian.

If `hessian_file` is found in `geometry.in`, the Hessian is constructed using the data in `hessian.aims`, which is a binary file generated by geometry optimization calculations with FHI-aims (please see `distributed_hessian`). The user should not try to create such a file by hand.

Tag: `trust_radius` (geometry.in)

Usage: `trust_radius value`

Purpose: In `geometry.in`, allows to specify the initial trust radius value for the `trm` relaxation algorithm.
`value` is the initial value set for trust radius, in Å. Default: 0.2 Å.

This keyword is a significant exception – it is the only algorithmic keyword found in the `geometry.in` file. Conceptually, it would belong into `control.in`, but since it is written out as part of the `geometry.in.next_step` file which can be used to restart a structure relaxation, we keep it here.

The keyword specifies the initial value of the "trust radius" used to limit structure relaxation steps determined by the `bfgs` (synonymous with `trm`) relaxation algorithm.

The default is set to 0.2 Å. It will be overridden by the default value of `max_atomic_move`, which can be set in `control.in`.

Specifically, the trust radius is compared to combined step length DX as proposed by the `bfgs` / `trm` algorithm across all active coordinates X_l, i.e., all coordinates that are not explicitly constrained:

$$DX = \sqrt{\sum_{l=1}^{N_{\mathrm{coords}}} (X_l^{\mathrm{new}} - X_l^{\mathrm{prev}})^2}. \tag{3.29}$$

X_l^{new} are the new coordinates that were proposed by the `bfgs` / `trm` algorithm, starting from the last accepted coordinates X_l^{prev} for which the total energies and energy gradients were calculated. The active coordinates X_l include all N_{coords} unconstrained atomic positions (x_I, y_I, z_I, where I=1, ..., N_{atoms}) and, if applicable, all unconstrained latticed vector coordinates ($a_{i,x}, a_{i,y}, a_{i,z}$, where i=1, 2, 3).

Since the `trust_radius` will be dynamically adjusted during a structure optimization run, an allowed minimum value can be set using the `min_trust_radius` keyword in `control.in`. If the `trust_radius` attempts to fall below this minimum value, the computation will stop since the forces likely deviate from the underlying energy surface.

Tag: `symmetry_n_params` (geometry.in)

Usage: `symmetry_n_params n_total n_lv n_coords`

Purpose: In `geometry.in`, specifies the number of parameters to be optimized in a symmetry-constrained relaxation. n_total is the total number of parameters, n_lv is the number of parameters that define the lattice cell, n_coords is the number of parameters defining the fractional atomic positions.
n_lv + n_coords = n_total
Example for an orthorhombic cell and no parameters for the atomic positions:
`symmetry_n_params 3 3 0`

This keyword also serves as a flag, setting `use_symm_const_geo_relaxation` internally to `True`. If `n_lv` is 0, then no unit cell relaxation is done. For unit cell relaxations make sure to set `relax_unit_cell` to `full`.

Tag: `symmetry_params` (geometry.in)

Usage: `symmetry_params [list of variables for parameters]`

Purpose: In `geometry.in`, list all variables that are used as parameters separated by blanks. Always list the lattice parameters first. The number of variables used for parameters has to be equal to n_total specifies in `symmetry_n_params`. Example for an orthorhombic cell and no parameters for the atomic positions:

```
symmetry_params a b c
```

Tag: `symmetry_lv` (geometry.in)

Usage: `symmetry_lv x , y , z`

Purpose: Specifies the analytic form of the lattice vectors. Use exactly the same cell as given in `lattice_vector` (same order of lattice vectors, same orientation), and simply replace entries that are free to relax by their analytic expression in terms of parameters specified by `symmetry_params`. Example for an orthorhombic cell:

```
symmetry_lv a , 0 , 0
symmetry_lv 0 , b , 0
symmetry_lv 0 , 0 , c
```

Note the comma between the components of the lattice vectors.

Tag: `symmetry_frac` (geometry.in)

Usage: `symmetry_frac` n_1 , n_2 , n_3

Purpose: Specifies the analytic form of the *fractional* atomic positions. Use exactly the same form as given in `atom_frac` (same order atoms), and simply replace entries that are free to relax by their analytic expression in terms of parameters specified by `symmetry_params`.
Example:

```
symmetry_frac 0.0 , 0.0 , x1
symmetry_frac 1./2 , 1./2 , x1 + 1./2
```

Note the comma between the fractional coordinates of the atoms and that no species entry is needed.

Tags for general section of `control.in`:

Tag: `aggregated_energy_tolerance` (control.in)

Usage: `aggregated_energy_tolerance tolerance`

Purpose: Sets the energy amount by which the energy across an entire relaxation trajectory may ever go uphill, based on the lowest known energy so far.

`tolerance` is a positive real number, in eV. Default: $3{\cdot}10^{-3}$ eV.

Small uphill steps of a relaxation trajectory are allowed up to the keyword `energy_tolerance`, but a relaxation trajectory should never go uphill for an extended number of steps in small uphill increments. The keyword `aggregated_energy_tolerance` sets an overall cap for any accepted uphill steps across an entire relaxation trajectory. The default is much larger than the allowed `energy_tolerance` in a single step and should, in principle, never be breached. If the `aggregated_energy_tolerance` criterion triggers, please contact us.

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: `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 [64, 208]. 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_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. Otherwise, `.false.`

This flag allows to request an explicit analytical stress tensor computation for an otherwise explicit single-point calculation.

The calculation of the analytical stress is limited to LDA, GGA, Meta-GGA and hybrid functionals and is not possible with `load_balancing`. The vdW correction based on Hirshfeld partitioning (`vdw_correction_hirshfeld`) is included into the analytical stress tensor.

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. 4.8). 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^{-4}.

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^{-4}.

Tag: `distributed_hessian` (control.in)

Usage: `distributed_hessian flag`

Purpose: If `.false.`, each MPI task holds a complete copy of the Hessian matrix. If `.true.`, the Hessian matrix is distributed across tasks.

`flag` is a logical string, either `.true.` or `.false.`. Default: `.true.` if both MPI and ScaLAPACK are available, `.false.` otherwise.

This keyword is particularly useful when relaxing a large structure. Please note that it only works if FHI-aims is built with both MPI and ScaLAPACK. If `init_hess` `reciprocal_lattice` is found in `control.in`, distributed storage of the Hessian will be automatically turned off.

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: $1.5{\cdot}10^{-3}$ eV.

Small upward steps during relaxation may occur as a result of a slightly mis-guessed `bfgs` 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. However, be sure to check that the total energy does not go up across several successive steps in a relaxation run. For the `trm` optimizer, also see `harmonic_length_scale` .

Tag: `external_force` (geometry.in)

Usage: `external_force x y z`

Purpose: *Experimental* – Applies an external force to the atom previous to this keyword.

`x,y,z` are the force components in eV/Å applied to the atom in x, y, z direction.

When an external force is applied it is necessary to contstrain the relaxation of at least on other atom to avoid a constant shift of the geometry. Also the value has to be reasonably chosen. Tear apart geometries can result in very flat energy landscapes, which take a

large amount of time to optimize. Typically this feature should be used in cases where a small external force, e.g. a STM-tip is applied on an atomic layer and the geomtry response of this external force is of interest.

This feature is experimental since no extensive testing was done for it.

Tag: `external_pressure` (control.in)

Usage: `external_pressure value`

Purpose: *Experimental* – Applies external pressure to the unit cell.

`value` is the pressure in eV/Å^3 applied to the unit cell.

In the periodic case, it is possible to apply hydrostatic pressure to the unit cell. To actually see the effect of the external pressure, a unit cell relaxation is required (see `relax_geometry` and `relax_unit_cell`). The crystal is then relaxed with the external pressure added to the stress tensor.

This feature is experimental since no extensive testing was done for it.

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` [64, 208] 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. 4.8).

Tag: `force_correction` (control.in)

Usage: `force_correction flag`

Purpose: When computing the forces, determines whether or not to include the Hartree potential force correction term. Consideration of this keyword can be helpful to speed up processes such as geometry relaxations or a molecular dynamics run.

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

As described elsewere[24, 48], omission of this term is one of the main reasons why the Hellmann-Feynman forces require a high level of self-consistency before their values can

be trustworthy. In fact, this term is only equal to zero at full self-consistency.

Because this correction is only meaningful at a low level of self-consistency, for an appropriate use of this keyword, sc_accuracy_rho must also be set within a reasonable value, i.e., a too tight threshold for the density can lead to an insignificant correction to the forces. Therefore, unless specified by the user, FHI-aims lowers the default value of sc_accuracy_rho accordingly if a calculation involves force_correction .

Although force_correction and sc_accuracy_forces can be used together inside a particular calculation, its joint use is strongly discouraged due to the considerably high cost that each forces computation imply.

It should be noted that currently, in case of relax_unit_cell or any kind of analytical stress computation, as well as usage of ouput , force_correction becomes ineffective in determining the default value of sc_accuracy_rho .

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 qualitiy 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 $-\Delta\tilde{E} = (\boldsymbol{X}_2 - \boldsymbol{X}_1)\cdot(\boldsymbol{F}_2 + \boldsymbol{F}_1)/2$ 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 or hessian_file .
flag is a logical string, either .true. or .false. Default: .true.

Note: The geometry.in.next_step file is written out by default when the relax_geometry bfgs algorithm is used.

Tag: `init_hess_lv_diag` (control.in)

Usage: `init_hess_lv_diag value`

Purpose: In a geometry relaxation with a unit cell optimization, allows to specify the initial Hessian matrix elements used to estimate relaxation steps that are associated with the lattice vector degrees of freedom.

`length` is a length scale in eV/Å^2. Default: 25 ev/Å^2

See the `init_hess` keyword for more information on the initial Hessian matrix used during a structure optimization.

Tag: `init_hess` (control.in)

Usage: `init_hess type [value]`

Purpose: Defines the initial Hessian matrix that is used by the `bfgs` structure relaxation algorithms (synonymous with `trm`) of the `relax_geometry` keyword.

`type`: Presently supported options are `diag [value]`, `Lindh [value]`.

Default: `init_hess Lindh 2.` unless a specific Hessian is given in the `geometry.in` file.

If no explicit initial Hessian matrix is given in the geometry.in file, the keyword `init_hess` can be used to specify the initial Hessian matrix used in a structure optimization.

With the option `diag`, a diagonal initial Hessian matrix is assumed. Then, the number given by the `value` option sets the diagonal elements between all atomic coordinates directly. The default is 25 eV/Å^2. Larger values lead to a more conservative start, smaller values lead to more aggressive initial relaxation steps.

With `Lindh` the Lindh model matrix [155] is used to initialize the Hessian between all *atomic* coordinates (usually a very efficient guess). For stability reasons, `add-value` (in eV/Å^2, defaults to 2.0 eV/Å^2) is added to all matrix elements on the diagonal. The parameter `thres` (default: 15.0) can be used to specify the accuracy of the Lindh matrix; only terms estimated to be larger than $e^{-\texttt{thres}}$ are taken into account. If you are planning a large number of complex unit cell optimizations of a similar type, we do recommend to check whether this default value is any good.

If a unit cell relaxation is additionally requested, the Hessian for the lattice degrees of freedom is set to be the square of the reciprocal lattice matrix and normalized to 25 eV/Å^2. This mimics a diagonal initial Hessian if strain coordinates for the representation of the lattice would be used. [192]

If the initial Hessian is specified explicitly by `hessian_block` or `hessian_file` in `geometry.in`, this explicit hessian overrides any information requested by the `init_hess` keyword.

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-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_trust_radius (control.in)

Usage: min_trust_radius value

Purpose: The minimum trust radius value that the relaxation algorithm is allowed to reach before aborting the structure optimization since no path forward can be found.

value is the allowed minimum trust radius in Å. Default: 10^{-4} Å.

In FHI-aims' standard relaxation algorithm, relax_geometry trm or bfgs, the trust_radius is the limit up to which a geometry optimization step is trusted and therefore allowed. The trust_radius fluctuates during execution. It can be reduced or it can increase, depending on the success or failure of consecutive geometry optimization steps. Notably, if the relaxation algorithm encounters a scenario in which it cannot find an admissible step, the trust_radius could, in principle decrease to arbitrarily low values, never making progress while spending significant computational resources.

The min_trust_radius value sets the lower bound that is considered acceptable for the trust_radius . If the trust_radius decreases below this bound, the relaxation algorithm concludes that there does not appear to be a sensible pathway forward and stops. This can happen, for example, if the calculated energy gradients ("forces") are inconsistent with the actual energy surface, i.e., if the forces point in a direction in which the actual energy goes uphill. Several possible strategies exist to escape this scenario – typically, denser integration grids or, perhaps, a tighter scf convergence criterion sc_accuracy_rho . However, it is also possible that a particular total energy method (e.g., a new development) does not yet have entirely consistent forces. Obviously, we'd like to know about such cases. In that case, contacting the developers would be helpful and appreciated.

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 reorthnormalize 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 and trm (synonymous with bfgs), as well as trm_2012 for reference with older FHI-aims versions.

- bfgs or trm is the recommended default. It uses a trust radius method enhanced version of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm (see Ref. [176], 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.

  As of December 2018, it additionally implements preconditioning of the lattice-lattice Hessian as explained in init_hess and preserves fractional coordinates of the atomic positions when predicting a new lattice. This mimics the optimization in strain coordinates. [192]

- trm_2012 Implements the former trm method *without* effective strain coordinates.

A reliable force convergence criterion tolerance for most structures is 10^{-2} eV/Å or $5{\cdot}10^{-3}$ eV/Å. **Do not use significantly smaller values unless you have a specific reason. Smaller values may cost much computer time for essentially no further measurable total energy minimization.**

Going to a much smaller tolerance value may only be useful for some very specific purposes, for example, high-accuracy finite difference calculations for vibrational properties. In other scenarios, if tolerance is set to a too small value by default, 80% or more of your CPU time may be spent groping around in the last meV of the structure optimization.

If tighter settings of the tolerance parameter are used, do not forget that tighter s.c.f. convergence accuracy settings may also be required to get accurate gradients in the first place. Ideally, use the sc_accuracy_rho keyword for this purpose, not sc_accuracy_forces or sc_accuracy_stress (see below).

In other words, use the tolerance criterion for a structure relaxation run wisely – decide what is the physical quantity you are actually interested in, and then check which value of the tolerance criterion is safe but still efficient.

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 [155] with a slight modification. If, for some reason, a particular initial geometry does not appear to play well with the Lindh Hessian, a simpler, slower, but more resilient diagonal approximation to the initial Hessian matrix can also be used. For more information see the init_hess keyword above.

The energy_tolerance and harmonic_length_scale keywords can be set to more forgiving values if the bfgs algorithm decides to abort relaxations because of a presumed deviation between the predicted and the real energy landscape.

Another important warning: Evaluating the forces and the stress tensor is **much** more expensive than a normal iteration during s.c.f. convergence. The current default

behavior of FHI-aims avoids any double computations of forces and stress tensors, relying instead on a sufficiently tight convergence criterion sc_accuracy_rho to determine s.c.f. convergence and only then calculating forces and stresses *once* per geometry step.

While one can in principle check the s.c.f convergence of forces / stresses explicitly, the cost of multiple evaluations of forces / stresses for a single geometry can be **very** high. Therefore, we recommend to *never* use the keywords sc_accuracy_forces or sc_accuracy_stress in a `control.in` file unless there is a specific need. Do not set these keywords routinely.

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`, `fixed_angles`. 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 affectd by explicit constraints.
- `fixed_angles` : All angles between lattice vectors will be constrained (kept fixed), only the lengths of each lattice vector are varied. (This option used to be called `shape`, but that is a misunderstandable term. The `shape` term will be removed in the future to avoid confusion.)

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, here are some potential remedies:

1. A very possible reason may be because the integration grids are not dense enough. This could especially well be the case for "light" settings. One remedy is to just use the integration grids from "tight" and the basis functions from "light".

2. Another possible remedy is to switch the way the integration weights are calculated to a slightly slower, non-default version. E.g., change the `partition_type` to a spherical one like `rho_r2`.

3. Finally, you may wish to set the convergence of the analytical stress with `sc_accuracy_stress` to an explicit, final value. **Only ever set this keyword for test purposes, though, not routinely. The cost for too many analytical stress evaluations can be disproportionately large.**

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: `write_restart_geometry` (control.in)

Usage: `write_restart_geometry flag`
Purpose: During a structure optimization, exports the current geometrys and approximation to the Hessian matrix to a file `geometry.in.next_step`.
`flag` is a logical string, either `.true.` or `.false.` Default: `.true.`

Note: The `geometry.in.next_step` file is written out by default when the `relax_geometry bfgs` algorithm is used.

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. [28]. For a truly thorough explanation of the underlying concepts, please refer to the standard literature, e.g., Refs. [70, 29].

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.* [142], 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 $\mathbf{c}_{\mathrm{new}} = (\mathbf{PS})_{\mathrm{extra}}\mathbf{c}_{\mathbf{old}}$. As a last step, the one-particle coefficients are orthonormalized.

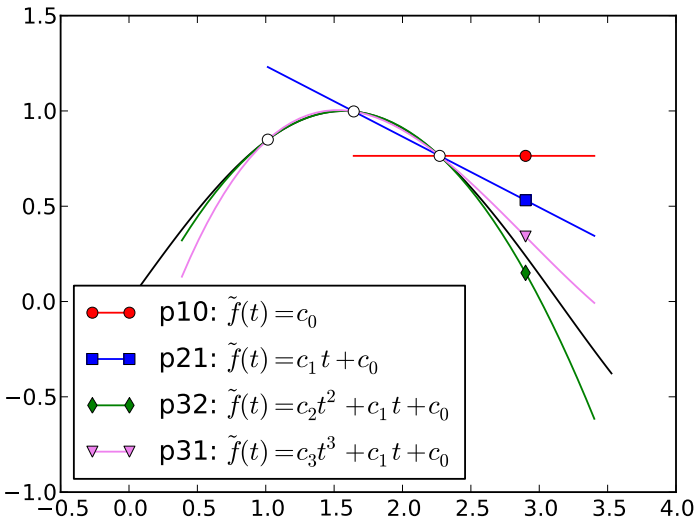


Figure 3.1: Different extrapolation schemes. The scheme "pno" refers to an n point scheme of order o, remaining orders used to fit odd components.

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 $f(t_0 - k\Delta t)$, $k = 1, \ldots, n$ 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 intialize 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 stemps Δt. 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 [136].

Therefore, in contrast to [194, 102], 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 a velocity for the immediately preceding `atom` in file `geometry.in`.

`vx`, `vy`, `vz` : x, y, and z components of the velocities, in Å/ps.

In `geometry.in`, the line containing the velocity must follow the line containing the `atom` that the velocity refers to. This can be used, e.g., to initialize a molecular dynamics run, or for analysis purposes later. Note that, for molecular dynamics, the FHI-aims standard output prints this information in the proper format, as part of a particular geometry associated with a molecular dynamics trajectory.

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 ist 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 `amu*bohr^2` (mass-oriented specification) or `cm^-1` (frequency oriented specification, to allow to connect the thermostat mass to characteristic frequencies of the system). Default: `amu*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.* [175]), `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` ω `unit`" ($\sin \omega t$), and "`cos` ω `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. [117]. 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 [5] 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. [38]

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. [38] 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. [38] to be practically independent of the value of τ, 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. [38]) 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 [43, 46, 44].

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 [43, 46, 44] 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://epfl-cosmo.github.io/gle4md/. 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://epfl-cosmo.github.io/gle4md/.

3.12.1 Path integral molecular dynamics and advanced types of dynamics

The best way to perform path integral molecular dynamics and other more advanced dynamics techniques in FHI-aims is through the i-PI python wrapper [45]. This code is available free of charge and can be downloaded from http://epfl-cosmo.github.io/gle4md/index.html?page=ipi. Information about the code can be found in http://ipi-code.org/ and we provide a quick tutorial on how one can make it work with FHI-aims in the tutorials folder within aimsfiles. Through this interface, one can perform all types of dynamics (classical or quantum) with a wide range of thermostats, barostats, and different path integral acceleration techniques. i-PI uses internet sockets for the communication with the client codes, making it also easy to join different codes in the same simulation (e.g. a thermodynamic integration), as long as they can all communicate with i-PI. Note that NPT and NST (constant stress) simulations are currently supported only for functionals where the analytical stress tensor is available. If you wish to perform an NPT simulation, please add also compute_analytical_stress .true. to your control.in file.

The basic keywords that can appear in the control.in file of FHI-aims are listed below. Examples of how to run FHI-aims bound to i-PI are available in the folder aimsfiles/examples/ipi with the corresponding i-PI and FHI-aims input files, as well as a short explanation on how to run the programs. Examples of FHI-aims being used with i-PI can also be found in the i-PI distribution. Please refer also to the i-PI manual and contact the developer responsible for the FHI-aims interface (Mariana Rossi) if you have any doubts about the usage of FHI-aims with this code.

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 a host name or an IP address. If you want to use UNIX sockets, add 'UNIX:' before the host address.

portnumber should contain the number of the port that the wrapper is listening to.

Tag: communicate_pimd_wrapper (control.in)

Usage: communicate_pimd_wrapper quantity

Purpose: Communicates the specified quantity to the i-PI code. Currently i-PI keeps track of this specified quantity for each step of the dynamics and each replica of the system (should they exist). They are written on a file created by i-PI that can be easily parsed for postprocessing purposes. The current available options are: dipole, polarizability,hirshfeld, workfunction and friction.

3.12.2 Running FHI-aims with i-PI over TCP/IP Sockets

Using TCP/IP sockets (Internet sockets) is the most straightforward and consistent way of using the i-PI wrapper with FHI-aims, particularly on HPC systems where UNIX sockets are unavailable. However, IP/TCP ports are assigned dynamically on a system, and therefore there can be no guarantee that a previously assigned port in an input file will be free when the calculation starts. To avoid this problem we recommend using the utility script *get_free_port.py* in the *utilities* directory to set the port to right before initializing the calculation either locally or inside the submission script on an HPC system. This script will automatically check if a requested port and change it if it is not free or find a free port, and update both the relevant information in the i-PI and FHI-aims input files. To use the script simply, run

```
> get_free_port.py -x inputs.template.xml \
                   -ox inputs.xml \
                   -c control.in \
                   -oc {system_name_descriptor}/control.in
```

or for cases when multiple calculators are used

```
> get_free_port.py -x inputs.template.xml \
                   -ox inputs.xml \
```

```
                    -c control.in \
                    -oc {system_1_name_descriptor}/control.in \
                        {system_2_name_descriptor}/control.in
```

In cases where you simply want to overwrite the template input files do not set the -ox or -oc variables. Additionally if you are running a system with a variable IP address, you can update the `hostaddress` with the following command:

```
> get_free_port.py --host HOST_ADDRESS \
                   -x inputs.template.xml \
                   -ox inputs.xml \
                   -c control.in \
                   -oc {system_name_descriptor}/control.in
```

where `--host default` will set the host to be the default for the system you are on. For a complete description of the functionality of the script run

```
> get_free_port.py --help
```

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 [234, 84], 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:

$$F(T,V) = F^{el}(V) + F^{nu}(T,V) = F^{el}(V) + F^{qh}(T,V) + F^{ah}(T,V)\ . \tag{3.30}$$

$F^{el}(T,V)$ is the free energy of the electronic system, which can be assessed by *Mermin's canonical generalization* of DFT [166]. 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 equibrium positions $\vec{R}_0$ on the approximative harmonic potential

$$V^{qh}(\vec{R}) \approx V^{nu}(\vec{R}_0) + \frac{1}{2} \sum_{\substack{L,\alpha \\ N,M,\beta}} \left. \frac{\partial^2 V^{nu}(\vec{R})}{\partial \left(R_{0,L}\right)_\alpha \ \partial \left(R_{N,M}\right)_\beta} \right|_{\vec{R}=\vec{R}_0} (U_{0,L})_\alpha (U_{N,M})_\beta\ . \tag{3.31}$$

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

$$F^{ah}(T,V) = F^{nu}(T,V) - F^{qh}(T,V)\ . \tag{3.32}$$

For this purpose, the dynamics of the *hybrid* system that is characterized by the potential

$$V^{\lambda}(\vec{R},\lambda) = \lambda\ V^{nu}(\vec{R}) + (1-\lambda)\ V^{qh}(\vec{R}) \tag{3.33}$$

is inspected. The parameter λ 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

$$F^{ah}(T,V) = \int_0^1 d\lambda \left(\frac{\partial F^{\lambda}(T,V,\lambda)}{\partial \lambda} \right)\ , \tag{3.34}$$

as the fundamental theorem of differential and integral calculus shows. The relation [234]

$$\frac{\partial F^{\lambda}(T,V,\lambda)}{\partial \lambda} = \left\langle \frac{\partial}{\partial \lambda} V^{\lambda}(\vec{R},\lambda) \right\rangle_{V_\lambda} \tag{3.35}$$

allows to replace the integrand in Eq. (3.34) 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 [234]

$$F^{ah}(T,V) = \int_0^t dt' \, \frac{d\lambda}{dt'} \left(\frac{\partial}{\partial \lambda} V^{\lambda}(\vec{R},\lambda) \right) \; . \tag{3.36}$$

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 λ 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` λ_{start} λ_{end} `QH_filename` `V0`

Purpose: Specifies the thermodynamic integration parameters for the immediately preceding `MD_segment` in file `control.in`.

λ_{start}, λ_{start} : Initial and final value for λ 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. [22, 23], 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.

For the purpose of simulating electronic excitations, either from electronic core levels (XPS) or from valence levels (UPS), the keywords `force_occupation_basis` and `force_occupation_projector` enforce electron occupations of specific atomic basis states or Kohn-Sham states, respectively.

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_region` s, 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 (for a short how-to cf. `force_occupation_projector`). 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 number of the highest energy Kohn-Sham state in the system that will be included in the constraint.

Example:

```
force_occupation_basis 1 1 atomic 2 1 1 1.3333 6
```

This choice will constrain the overall occupation of a given basis function (not Kohn-

Sham state!) in the system to 1.3333 electrons.

The basis function in question resides on the first atom (number 1) as listed in `geometry.in`. The first spin channel is constrained.

Since we are interested in constraining an atomic-like orbital, we choose one that is part of the minimal basis (type "atomic").

In fact, we let the constraint act on a $2p$ level, m=1 (defined by the sequence "2 1 1").

Only Kohn-Sham orbitals up to the 6th state (counted in the overall system) will be included in the constraint. In general, it is a good idea to constrain the orbital in question out of the occupied space, i.e., choose a value for `max_KS_state` that indicates a state above the Fermi level.

There is a small problem here: We need to define the occupation of a "basis function," but really, we here have a non-orthogonal basis. Strictly speaking, only the Kohn-Sham states in the system have a well-defined occupation. What to do?

One thing we could use (and we do) are Mulliken occupation numbers of the basis functions. These are formally always well defined. In practice, however, as the overall basis set becomes larger and larger and approaches (over-)completeness, Mulliken occupations become less and less meaningful because other basis functions are not exactly orthogonal to the one used in our projection, and can "restore" the component that was originally constrained away.

Either way, we use Mulliken occupations, assuming that the atomic core basis functions are sufficiently compact and practically orthogonal to everything else.

This assumption will work well for localized basis functions such as the $1s$ levels of most elements. As a rule, the constraint is expected to be less and less unique if applied to more delocalized basis functions – there can even be multiple different self-consistent constrained solutions for the same formal constraint. For instance, Si $2p$ basis functions can exhibit this problem if the basis sets on the surrounding atoms – which overlap with the Si atom – become too large. Here, a smaller basis set (tier 1) can indeed be the way to keep a qualitatively meaningful Mulliken-type constraint.

Tag: `force_occupation_projector` (control.in)

Usage: `force_occupation_projector KS_state spin occ KS_start KS_stop`

Purpose: This keyword enforces the occupation `occ` in `KS_state` of spin channel `spin`. Between different SCF steps the overlap of this state with states `KS_start` to `KS_stop` is being checked and the constraint is changed correspondingly if the main character of the state changes.

Example:

```
force_occupation_projector 8 1 0.0 6 10
force_occupation_projector 9 2 1.0 6 10
```

This enforces 0.0 occupation in state 8 of spin channel 1 and 1.0 occupation for state 9 of spin channel 2. KS states between 6 and 10 are checked for overlap with state 8 and 9 of previous SCF steps. If 8/1 was occupied and 9/2 was unoccupied in the ground state, this corresponds to a triplet excitation 8→9.

To simulate XPS energies with n inequivalent atoms of the same species (called excitation centre in the following) a total of $n + 1$ single runs is required: One ground state calculation and one `force_occupation_basis` / `force_occupation_projector` calculation for each of the excitation centres.

The ground state calculation should use the `restart_write_only` or the `restart` flag to create a restart file that is needed for the `force_occupation_basis` or alternatively the `force_occupation_projector` run. Therefore the geometry.in and all other parameters (except the charge) such as basis sets have to match. In practise it is often beneficial for the interpretation of the XP spectra to use `output cube eigenstate` to have an idea of the localization of the different core levels.

For the simulation of the XPS spectra typically a full core hole is introduced at the excitation centre (for example in ref. [58]). Relying on initial state effects alone (i.e., defining the ionization energies using ground state eigenvalues) neglects the screening of the core hole by valence electrons [156] and is therefore not a good approximation for XPS. For example, to force the occupation of eigenstate 3 to 1.0, where states 1-4 are (near-)degenerate or at least very similar in energy and type:

```
force_occupation_projector 3 1 1.0 1 4
```

Results in the following occupation (the introduction of the core hole leads to a re-ordering):

```
  State  Occupation  Eigenvalue [Ha]  Eigenvalue [eV]
    1      1.00000      -16.148150       -439.41353
    2      2.00000      -14.162982       -385.39434
    3      2.00000      -14.162981       -385.39432
    4      2.00000      -14.091396       -383.44640
```

Note that `charge` was set to 1 to take into account the reduced electron number and a restart from the ground state run was made using `restart` .

XPS energies can then be calculated as the difference of the total energy obtained in the ground state calculation and the total energy of the core-hole excited simulation (corresponding to the definition of the ionization energy). That means that for each excitation center an ionization energy is calculated. For the core level shifts only relative energy differences are relevant, which are already directly reflected in the differences of total energies of the core-hole excited states. If, however, absolute energies are of interest, note that experiments are referenced to either the vacuum level or the Fermi level, and that simulations including an extended surface might differ by the workfunction from those for isolated molecules. The ionization energies can then by broadened with Gaussian functions of same amplitude (assuming no preferential direction, especially valid for $1s$ spectra) and summed up to obtain the total XPS spectrum.

Simulating NEXAFS spectra can be less straightforward as there are different approximations to account for the core hole and the excited electron. One possibility is to use the transition potential approach [229], where instead of a full core only half a core hole is used, i.e., $n = 0.5$ in one spin channel. Independently from the approximation used for the core hole: To obtain the dipole matrix elements that give information about the transition probability the flag `compute_dipolematrix` needs to be used. Note that to use this option the FHI-aims binary has to be compiled enabling hdf5, as the output is a hdf5 container containing eigenvalues and matrix elements. It is recommended to include additional `empty_states` , depending on the amount of unoccupied states you want to probe. In this case the ground state calculation has already to include the same number of empty states, otherwise a restart is not possible.

Tag: `force_occupation_smearing` (control.in)

Usage: `force_occupation_smearing smearing_width`

Purpose: If keyword is set, the occupation constraints are enforced in form of gaussians with a width of `smearing_width` instead of delta peaks. This applies to orbitals within an energy range of -+ 3*`smearing_width`

Default: No Smearing at all.

Example:

```
force_occupation_smearing 0.05
```

This keyword helps to converge systems with state degeneracies, which are constrained by force_occupation_projector. Specifically when calculating electronic excited states in the frontier orbital regime, many state degeneracies can occur. If one of two degenerate states is constrained to a different than the ground state occupation, convergence can be hindered. If this happens, this keyword can enable convergence for the price of a minimally different final occupation and therefore also a small error in excitation energy. One should be very careful with this keyword and only employ it if convergence can not be reached without.

WARNING: It is very easy to generate reandom numbers when using this keyword. The smearing value should never exceed 0.15

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.

Warning: When using a multipole, e.g., an external charge with no basis functions etc., you are creating a Coulomb singularity. If this singularity is inside the radius of a basis function of another atom, it will lead to numerical noise in integrals, up to near-infinities.

To test and/or overcome this problem, all you need to do is to place an integration grid on any multipole that is within the radius of a basis function of any atom. This radius is given by the cutoff radius plus width in the `cut_pot` keyword of each `species`.

Such a grid can be placed by creating an `empty` site with no basis functions (`include_min_basis` .false.) and placing this empty site on the same site as the multipole in question in `geometry.in`. Simply taking the species defaults for a H atom (light settings) and adjusting them to have no basis functions should create the necessary definition of the empty site in question (see Fig.3.2 for an example).

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 $\boldsymbol{E}$.

`E_x` is a real number, the x component of $\boldsymbol{E}$ in V/Å.
`E_y` is a real number, the y component of $\boldsymbol{E}$ in V/Å.
`E_z` is a real number, the z component of $\boldsymbol{E}$ in V/Å.

Please note: The electrical field is usually defined to point in the direction of a force exerted on a *positive* probe charge. Historically grown, FHIaims uses the opposite sign convention. Although this behaviour might be considered a bug, we decided to leave it this way in order not to break any scripts people are already using.

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·Å.

Warning: Note that monopoles amount to Coulomb singularities. When inside the basis function radius of any atom, such monopoles should be covered with an integration grid in `geometry.in`, as explained in the beginning of this section.

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 [212] that uses the energies and forces provided by FHI-aims as a "plugin".

```
[...]
  species        empty_site
#     global species definitions
    nucleus             1
    mass                1.00794
#
    l_hartree           4
#
    cut_pot             3.5  1.5  1.0
    basis_dep_cutoff    1e-4
#
    include_min_basis .false.

    radial_base         24 5.0
    radial_multiplier   1
    angular_grids       specified
      division   0.2421   50
      division   0.3822  110
      division   0.4799  194
      division   0.5341  302
      outer_grid  302
################################################################################
#
#  Definition of "minimal" basis
#
################################################################################
#     valence basis states
    valence      1  s   1.
#     ion occupancy
    ion_occ      1  s   0.5
################################################################################




[...]
```

Figure 3.2: Species data for what could be used as an empty site on top of monopole.

3.16 QM/MM Embedding

Please continue to consider this functionality 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 [131]. Those pseudopotential files can be either generated manually with the open source program package FHI98PP [75] or downloaded from `http://www.abinit.org/downloads/psp-links/lda_fhi` or `http://www.abinit.org/downloads/psp-links/gga_fhi`.

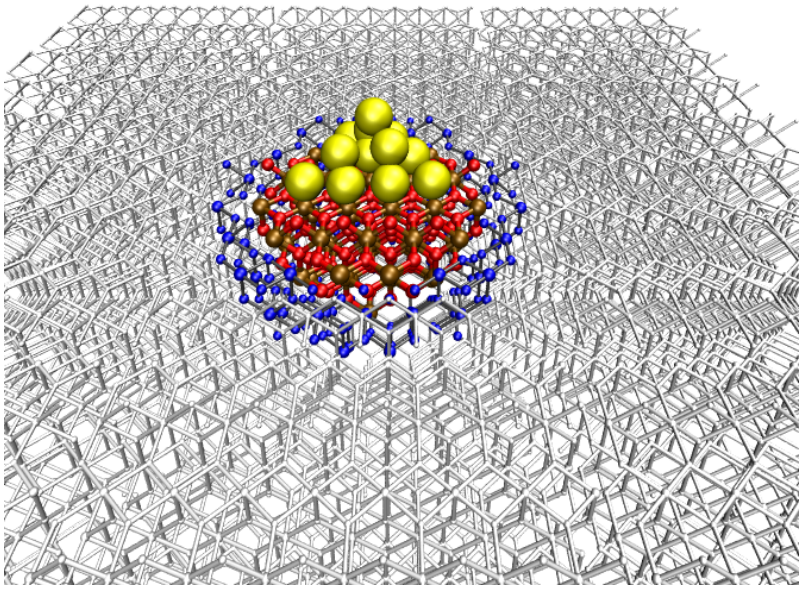


Figure 3.3: Example for QM/MM setup: $Au_n@TiO_2$. The adsorbed cluster and direct substrate vicinity defines the QM-region. The far field surrounding (grey particles) pictures a monopole field with formal charges (4+ for Ti and 2- for O). In the blue region, oxygen particles are still represented as 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`. Accept 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 in 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 [75]. 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 [75].

value: integer value

The choice which l-channel should be the local component is essential for the performance of the pseudopotentials. Again, read [75] for further help.

---

species sub-tag: nonlinear_core (control.in)

Usage: nonlinear_core flag

Purpose: 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 [157].

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 3.4: 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 Continuum Solvation Methods

The simulation of a solvent in a quantum mechanical calculation can, in principle, be done in two ways. One way is to include explicit solvent molecules in the calculation. This straightforward approach usually requires molecular dynamics (MD) simulations in order to yield thermodynamically meaningful observables as e.g. solvation free energies.

The second way is to average the effect of the solvent and treat it as a continuum which responds to the electrostatic potential created by the solute, i.e. the entity that is to be solvated. There are several flavors to this comparatively inexpensive approximation, e.g. the polarizable continuum model (PCM) [165], the conductor like screening model (COSMO) [130], the self-consistent continuum solvation (SCCS) model [6], the "SMx" models [53, 161, 162], or CMIRSv1.1 [242] to name some of the more popular ones. Statistical sampling then only needs to be performed for the degrees of freedom of the solute which obviously makes it computationally much cheaper.

In general, the necessary integration of the solvent's degrees of freedom beforehand leads to a problem where one now needs to solve a generalized Poisson's equation,

$$\boldsymbol{\nabla}\left(\varepsilon_0\varepsilon(\mathbf{r})\boldsymbol{\nabla}\Phi(\mathbf{r})\right) = -4\pi\varrho(\mathbf{r}), \tag{3.37}$$

to obtain the electrostatic potential $\Phi(\mathbf{r})$ created by the total charge density $\varrho(\mathbf{r})$ which now accounts for the electrostatic polarization potential of the solvent (often called "reaction field"). Eq. 3.37 contains a spatially dependent dielectric permittivity function $\varepsilon(\mathbf{r})$ in contrast to the regular Poisson's equation,

$$\boldsymbol{\nabla}\left(\varepsilon_0\boldsymbol{\nabla}\Phi_{\mathrm{H}}(\mathbf{r})\right) = -4\pi\varrho(\mathbf{r}), \tag{3.38}$$

which is solved in a regular DFT calculation in every step of the SCF cycle to get the Hartree potential Φ_{H}. $\varepsilon(\mathbf{r})$ is defined in such a way that it has the experimentally accessible bulk value of the solvent at large distance to the solute. Inside the region in space occupied by the solute, the so-called 'cavity', it has a value of 1, corresponding to vacuum permittivity. Any polarization inside that region is assumed to be accounted for at the DFT level. The definition of the cavity and the transition of $\varepsilon(\mathbf{r})$ at its boundary differ between different implicit solvation models.

Currently, FHI-aims supports two models which have different strengths and capabilities which are summarized in Table 3.1. The Multipole Expansion (MPE) implicit solvation method uses a sharp, step-like transition of $\varepsilon(\mathbf{r})$ at the cavity boundary. It separates the FHI-aims grid into two (original MPE method[215]) or multiple (MPE-nc[69]) domains and couples them via electrostatic boundary conditions. It therefore in fact solves multiple coupled Poisson equations with different dielectric permittivities. The Finite ion-size and Stern layer modified Poisson-Boltzmann (SMPB) method solves a single Poisson equation on the full FHI-aims integration grid by defining a smooth dielectric permittivity function. In general, the accuracy of the evaluation of solvation energies is expected to be similar. In fact, both methods merge into each other if the dielectric transition in the SMPB model is turned into a sharp step function. On top of this ion-free implicit solvation model, the SMPB approach also supports the modeling of finite ionic strengths in the solution.

The MPE solvation model is the faster one of both approaches with only a small overhead with respect to vacuum calculations. The overhead of both implicit solvation methods is reduced, when performing expensive hybrid calculations, since the actual time for the implicit solvation calculations does not vary with the functional.

	MPE	**SMPB**
Solvent Parametrizations	H_2O (N,C)	H_2O (N,C), CH_3OH (N,C) (more in work)
Dissolved ions/salt	no	yes (SMPB/LPB)
Salt Parametrizations	–	Aq. Monoval. Salt Solutions
CPU speed	fast	moderate
Forces	no	yes
PBCs	no	no (in work)
Developers	Markus Sinstein Jakob Filser	Stefan Ringe Christoph Muschielok, Marvin H. Lechner

Table 3.1: Comparison of the two implicit solvation methods in FHI-aims. Parameter sets for the single cavity MPE method are available in ref. [215], for MPE-nc in ref. [69], for the SMPB method in ref. [6] and [63] (parameters for methanol as more solvents in current work). N and C indicate parameter sets fitted for neutral and charged solutes, respectively.

In the following, both models are summarized and the key input parameters presented.

3.17.1 MPE Implicit Solvent Model

This functionality is not yet available for periodic systems.

The current implementation does not have analytical forces yet.

Generally, when combining MPE with other functionality, you should know what you are doing. No specific interactions with other methods beyond single point DFT are implemented, so only methods which do not interfere with MPE are safe to use.

Default behaviour: If `solvent mpe-nc` is set, an MPE-nc calculation in water will be performed with *tight* expansion orders and isodensity and non-electrostatic parameter optimized for the respective `xc` functional (or optimized for HSE06, if no parameterization for the chosen functional exists), see ref. [69], table 2. If `solvent mpe2017` is set and the solute charge is 0 or +1, then the solvation parameters for neutrals and cations (SPANC) are used.[215]

As outlined in more detail in Ref. [215], the multipole expansion (MPE) implicit solvent model offers an efficient way of solving Eq. 3.37 based on the knowledge of the Hartree potential Φ_H readily available from a splined representation in FHI-aims (cf. Sec. 3.7) via least-squares fitting instead of integration. The central Ansatz is

$$\Phi(\mathbf{r}) = \varepsilon^{-1}(\mathbf{r})\Phi_H(\mathbf{r}) + \Phi_{MPE}(\mathbf{r}) \tag{3.39}$$

With the potential that the solute would generate in vacuum $\Phi_{\mathrm{H}}(\mathbf{r})$ and the inverse permittivity $\varepsilon^{-1}(\mathbf{r})$. The remaining part $\Phi_{\mathrm{MPE}}(\mathbf{r})$ of the potential is defined separately for the cavity and the dielectric region. The dielectric function $\varepsilon(\mathbf{r})$ here needs to be a step-function in 3D-space where the following boundary conditions apply at the step ($\mathbf{n}$ denotes the normal direction to the interface):

$$\Phi_+ = \Phi_- \tag{3.40a}$$

$$\mathbf{n} \cdot \varepsilon_+ \boldsymbol{\nabla} \Phi_+ = \mathbf{n} \cdot \varepsilon_- \boldsymbol{\nabla} \Phi_- \tag{3.40b}$$

Then, the above equations are discretized in two ways:

- The potentials Φ_{MPE} in- and outside the cavity are expressed in a truncated multipole series with expansion orders $l_{\mathrm{max,R}}$ and $l_{\mathrm{max,O}}$, and
- equations 3.40a and 3.40b are evaluated at N points on the interface manifold.

Thereby, N is chosen such that the resulting system of linear equations (SLE) is overdetermined (typically by a factor of two to three).

MPE-nc additionally separates the cavity into atom-centered subregions, each with its own multipole series. This ensures fast convergence of the total (DFT) energy with expansion order. In contrast to the earlier single-cavity version, it does not rely on a specific kind of solute size dependent error cancellation, improving transferability (although different kinds of error cancellation are likely still present).[69] Although slightly more costly than the single-cavity version, MPE-nc is typically not the computational bottleneck of any calculation. It is therefore more advisable in most cases.

One noteable exception may be small (fewer than ≈ 10 non-hydrogen atoms) neutral or cationic solutes, for which a parameterization of single-cavity MPE with `isc_cavity_type rho_multipole_static` exists with an overall smaller reported error than any of the MPE-nc parameter sets.[215] Note that this cavity type was simply never parameterized for MPE-nc, because with this cavity definition, parameters are not transferable to anionic solutes. If desired, a parameterization of MPE-nc with `isc_cavity_type rho_multipole_static` for neutral and cationic solutes is certainly possible and would likely be at least equally as accurate as the single-cavity MPE version.

If needed for this special case, or for consistency with previous work, the original implementation can still be accessed by `solvent mpe2017`.

Tags for general subsection of `control.in`:

The keywords controlling the MPE module are divided into six categories:

elementary These are the most important keywords which define the physical model.

convergence Here, the most important convergence parameters are collected which should be checked before doing (large scale) production runs.

performance These settings should only influence computational time and memory requirements.

expert These settings should only be modified by an experienced user as they allow quite profound modifications.

debug Debug settings are intended to give valuable insight for developers into intermediate results.

deprecated These settings are no longer actively maintained and should not be used except by developers who know what is actually implemented.

elementary

Tag: `solvent` (control.in)

Usage: `solvent method`

Purpose: Specifies the desired implicit solvent model.

Options: `method` is a string which specifies the implicit solvent method

- `mpe-nc`: MPE-nc[69]
- `mpe2017`: Single cavity MPE.[215] Note that this is *not* entirely equivalent to the deprecated option `mpe`, as the defaults for `mpe_solvent_permittivity`, `isc_cavity_type` and `mpe_nonelectrostatic_model` have changed.
- `mpb` (cf. Sec. 3.17.2)

All further options described in this section apply to both `mpe-nc` and `mpe2017`. Different choices of `isc_cavity_type` and `mpe_nonelectrostatic_model` are required for the two models.

Tag: `mpe_solvent_permittivity` (control.in)

Usage: `mpe_solvent_permittivity epsilon`

Purpose: Specifies the dielectric constant of the bulk solvent.

`epsilon` is a positive real number equal to the macroscopic dielectric constant of the solvent.

Default: `78.36` (water)

Tag: `isc_cavity_type` (control.in)

Usage: `isc_cavity_type type`

Purpose: This keyword controls the model used to sample the implicit solvent cavity for the MPE method. Depending on `type`, further flags (or even lines) might be necessary. Those are explained below.

Options: Currently supported options are `rho_free`, `rho_multipole_static`, `overlapping_spheres`, and `rho_multipole_dynamic`. Parameterizations exist only for the first two options. The latter two are thus described in the expert section.

Default: If `solvent mpe-nc`, then `rho_free` with isodensity optimized for water and the chosen `xc` functional, cf. ref. [69], table 2.
If `solvent` mpe2017, and the solute charge is 0 or +1, then `rho_multipole_static` with the solvation parameters for neutrals and cations (SPANC).[215]

`isc_cavity_type` sub-tag: `rho_free` (control.in)

Usage: `isc_cavity_type rho_free rho_iso`

Purpose: Constructs the cavity as an iso-density surface of the superposed electron density of the neutral, free atoms in the solute.

`rho_iso` is a positive real number specifying the desired iso-density value in units of $e\,\text{Å}^{-3}$.

`isc_cavity_type` sub-tag: `rho_multipole_static` (control.in)

Usage: `isc_cavity_type rho_multipole_static rho_iso`

Purpose: Constructs the cavity as an iso-density surface of the (multipole-expanded) converged electron density of the solute in vacuum, keeping the cavity static throughout the actual MPE calculation.

First, a vacuum calculation (SCF with any MPE related keywords turned off) will be performed until self-consistency is achieved, then SCF will be restarted, with specified MPE related keywords turned on and the isodensity cavity constructed from the converged electron density in vacuum.

`rho_iso` is a positive real number specifying the desired iso-density value in units of $e\,\text{Å}^{-3}$.

Tag: `mpe_nonelectrostatic_model` (control.in)

Usage: `mpe_nonelectrostatic_model model`

Purpose: This keyword controls any additional, "non-electrostatic" term not included in the purely electrostatic treatment of the solvent.

Options:

- `linear_OV` : see below
- `none`: compute no non-electrostatic terms

Default: If `solvent mpe-nc`, then `linear_OV` with β=0 and α optimized for water and the chosen `xc` functional, cf. ref. [69], table 2.
If `solvent` mpe2017, and the solute charge is 0 or +1, then the solvation parameters for neutrals and cations (SPANC).[215]

`mpe_nonelectrostatic_model` sub-tag: `linear_OV` (control.in)

Usage: `mpe_nonelectrostatic_model linear_OV` α β

Purpose: Corrects the total energy term by $\alpha O + \beta V$ where O is the surface area of the cavity and V its volume.

α is a real number in units of $\mathrm{eV\,\AA^{-2}}$.
β is a real number in units of $\mathrm{eV\,\AA^{-3}}$.

This non-electrostatic model is in principle identical to the one proposed by Andreussi *et al.* [6]. Note, however, that the surface tension of the solvent is here included in the parameter α.

convergence

Tag: `mpe_lmax_rf` (control.in)

Usage: `mpe_lmax_rf lmax`

Purpose: Specifies the expansion order of the polarization potential aka reaction field inside the cavity.

`lmax` is a non-negative integer number.

Default: `8`

This is a critical convergence parameter of the MPE model. For MPE-nc, a value of 8 should be sufficent in most cases. 12 is a safe choice. If MPE ever becomes the bottleneck of the calculation (which usually isn't the case), then 6 can be used for a "quick and dirty" approximate calculation.[69]

`solvent` mpe2017 generally needs higher expansion orders for larger solutes, but also relies to some degree on error cancellation of an underconverged electrostatic energy with an overestimated non-electrostatic attraction, as discussed in ref. [69].

Tag: `mpe_lmax_ep` (control.in)

Usage: `mpe_lmax_ep lmax`

Purpose: Specifies the expansion order of the polarization potential aka reaction field outside of the cavity.

`lmax` is a non-negative integer number.

Default: With `solvent mpe-nc`, default is 6. With `solvent mpe2017`, default is maximum value of `l_hartree` for all species.

This parameter is similar to but usually less critical than `mpe_lmax_rf` . For MPE-nc, a value of 6 should be sufficent in most cases. 8 is a safe choice. If MPE ever becomes the bottleneck of the calculation (which usually isn't the case), then 4 can be used for a "quick and dirty" approximate calculation.[69]

Note that `mpe2017` places expansion centers for the external potentials on solute hydrogen atoms, whereas MPE-nc does not. Both place centers on non-hydrogen atoms.

Tag: `mpe_degree_of_determination` (control.in)

Usage: `mpe_degree_of_determination dod`

Purpose: Defines the desired ratio of number of rows to columns in left-hand side matrix of the MPE equation.

`dod` is a real number ≥ 1.0.

Default: `5.0`

For the very limited (!) number of applications of the MPE method so far, the default value of 5.0 has been a save choice. Note, that the requested degree of determination can only very approximately be reached. This can lead to an under- determination of the MPE equations and a subsequent termination of the program when values for `dod` very close to 1 are chosen.

performance

Tag: `mpe_solver` (control.in)

Usage: `mpe_solver type`

Purpose: Defines the numerical method used to solver the MPE equations.

Options: `type` can be chosen from: `direct`, `lsqr`.

Default: `direct`

The default behavior is to factorize the left hand side of the MPE equations using any of the methods described in `mpe_factorization_type` . This allows to robustly solve the MPE equation via the pseudo-inverse of the left-hand side.

`lsqr` uses the iterative LSQR algorithm with a sparse representation of the left hand side matrix. This only makes sense together with `solvent mpe-nc`, as there will be no sparsity otherwise. While the direct solvers are generally faster, LSQR requires less memory and can be used as a fallback option for very large systems. Note also the caveat at `mpe_factorization_type` .

Tag: `mpe_factorization_type` (control.in)

Usage: `mpe_factorization_type type`

Purpose: Defines the numerical method used to factorize the left-hand side of the MPE equations as the first step to the numerical solution. Has no effect if `mpe_solver lsqr` is chosen.

Options: `type` can be chosen from: `qr`, `qr+svd`, and `svd`.

Default: If `solvent mpe-nc` is set, and `mpe_lmax_rf` is 8 or smaller and `mpe_lmax_ep` is 6 or smaller, then the default is to perform a QR factorization and solve the linear system directly from there.
If `solvent mpe2017` is used, or if one or both of the two expansion orders is higher than the above values, the default behavior is to perform a QR factorization with a singular value decomposition (SVD) on top. This allows to robustly solve the MPE equation via the pseudo-inverse of the left-hand side. This is the safest option, and sometimes necessary to deal with ill-conditioning at high expansion orders.

`qr` is generally significantly faster than the other options. However:

Caveat! Using the (non rank-revealing) QR factorization alone can fail when the left-hand side is rank deficient which can easily happen—especially for large expansion orders `mpe_lmax_rf` and/or `mpe_lmax_ep` ! On the other hand, `svd` does not necessarily mean that no QR factorization is performed as this (at least for the parallel implementation) depends on the (Sca)LAPACK driver routine used.

Tag: `mpe_sc_block_size` (control.in)

Usage: `mpe_sc_block_size size`

Purpose: Defines the ScaLAPACK block size used for solving the linear system of MPE. This may have an influence on computational time in some cases.

`size` is a positive integer number. If chosen too large to distribute the matrix over the given number of processes, the size will be reduced automatically.

Default: With `mpe_solver direct`: 32
With `mpe_solver lsqr`: 16

Tag: `mpe_tol_load_imbalance` (control.in)

Usage: mpe_tol_load_imbalance tol

Purpose: Works only together with mpe_solver lsqr. Defines how uneven the matrix elements are allowed to be distributed over processes. Example: If the MPE matrix has 1000 non-zero elements, and you are running on 10 cores, then it would ideally be possible to give exactly 100 elements to each process. With mpe_tol_load_imbalance 1.05, each process can be given up to 105 elements.

tol is a real number > 1.

Default: 1.05

Tag: mpe_tol_overdistribution (control.in)

Usage: mpe_tol_overdistribution tol

Purpose: Works only together with mpe_solver lsqr. Defines the factor by which a matrix block can be distributed over more cores than would ideally be necessary. Example: If the MPE matrix has 1000 non-zero elements, and you are running on 10 cores, then a matrix block with 200 elements could ideally be distributed over 2 cores. With mpe_tol_overdistribution 4, it can be distributed over up to 8 cores.

tol is a real number > 1.

Default: 4.0

The load balancing of mpe_solver lsqr tries to distribute matrix blocks over processes in such a way that the maximum number of matrix elements on any process is minimized while not distributing each individual block over more processes than is necessary. If no compromise is found, both mpe_tol_load_imbalance and mpe_tol_overdistribution will automatically be gradually increased until one is found.

expert

---

isc_cavity_type **sub-tag: rho_multipole_dynamic** (control.in)

Usage: isc_cavity_type rho_multipole_dynamic rho_iso

Purpose: Constructs the cavity as an iso-density surface of the *self-consistent* (multipole-expanded) electron density of the solute, updating the cavity in each SCF step.

rho_iso is a positive real number specifying the desired iso-density value in units of $e\,\text{Å}^{-3}$.

With this method, the cavity is updated to the current electron density in every SCF step. This also means that the MPE equations have to be solved in every SCF step making it computationally more expensive.

Caveat: Although it should in principle be implemented, the combination of this option together with `solvent mpe-nc` is not being actively maintained.

`isc_cavity_type` sub-tag: `overlapping_spheres` (control.in)

Usage: `isc_cavity_type overlapping_spheres type value`

Purpose: Constructs the cavity as a superposition of overlapping spheres around all atoms.

`type` specifies how the radii of the atomic spheres are determined.

- `radius`: all spheres have the same radius given by `value` in units of Å;
- `rho`: the atomic spheres are iso-density surfaces based on the electron density of the isolated, neutral atom with an iso-value of `value` in units of $e\,\text{Å}^{-3}$.

`value` is a real number whose meaning and units depend on the choice of `radius` (see above).

WARNING: The usage of this cavity type is strongly discouraged! It has been helpful in the development to analyze the cavity sampling process itself. The resulting cavities, however, are almost certainly not smooth and were never intended to be used in production calculations. When used with the MPE model, the whole calculation is prone to numerical problems and the results are very often unphysical. Instead, use the `rho_free` type that builds the cavity based on the superposition of atomic densities (which is again smooth) or use other types based on the (self-consistent) electron density of the solute.

Tag: `mpe_tol_adjR2` (control.in)

Usage: `mpe_tol_adjR2 tol`

Purpose: Defines the tolerance for the adjusted coefficient of determination $\bar{R}^2$ of the solved MPE equations. Will abort if $\bar{R}^2 < 1 - tol$.

`tol` is a real number between 0.0 and 1.0. Default: `0.05`

The MPE equations are sometimes not solvable in the regular solid harmonic basis used for the reaction field. This is the case especially for large molecules. A low $\bar{R}^2$ indicates such a bad solution. In some cases increasing `mpe_lmax_rf` helps, but there are pathologic cases where increasing `mpe_lmax_rf` leads to a perpetual decrease in $\Delta^{\text{el}}_{\text{solv}}G$ without ever converging.

For $\bar{R}^2 < 0.95$ it is likely that less than 90% of $\Delta^{\text{el}}_{\text{solv}}G$ are captured. This is however based on experience from a limited number of cases. Feedback to the developers (filser@fhi-berlin.mpg.de) will be appreciated!

Tag: `mpe_tol_adjR2_wait_scf` (control.in)

Usage: `mpe_tol_adjR2_wait_scf bool`

Purpose: If `.true.`, will wait until the SCF cycle is converged before it is checked whether $\bar{R}^2 < 1 - tol$.

Default: `.false.`

Although MPE does not actively try to converge, $\bar{R}^2$ tends to improve during the SCF procedure. Setting `mpe_tol_adjR2_wait_scf` can thus help borderline cases converge, at the cost of spending the full computation time of the SCF procedure on a calculation that might ultimately fail.

Tag: `isc_surface_curvature_correction` (control.in)

Usage: `isc_surface_curvature_correction bool`

Purpose: When this flag is turned on, the calculated surface area (and volume) of the cavity is approximately corrected for the cavity curvature.

`bool` is of Boolean type. Default: `.true.`

The effect of this keyword is usually rather negligible. For more details regarding the correction, please consult Ref. [215].

Tag: `isc_rho_rel_deviation_threshold` (control.in)

Usage: `isc_rho_rel_deviation_threshold threshold`

Purpose: Defines the convergence criterion of the cavity generation process: The walker dynamics simulation is run until the density values for all walkers deviate from the chosen iso value by at most `threshold`.

`threshold` is a small, positive real number. Default: 1×10^{-3}

This keyword is only applicable for an `isc_cavity_type` defined by an iso-density value.

Tag: `isc_max_dyn_steps` (control.in)

Usage: `isc_max_dyn_steps num`

Purpose: Determines the maximum number of allowed steps to reach convergence of the walker dynamics simulation in the cavity creation process.

`num` is a positive integer number. Default: 300

Tag: `isc_try_restore_convergence` (control.in)

Usage: isc_try_restore_convergence bool

Purpose: When convergence of the cavity creation dynamics run could not be achieved within the number of allowed steps specified by isc_max_dyn_steps , this flag allows to enforce convergence by simply deleting all walkers not satisfying the convergence criterion given by isc_rho_rel_deviation_threshold .

bool is of Boolean type. Default: .false.

Although a simple check is done to stop the calculation when too many walkers do not satisfy the convergence criterion, one should always manually checking the resulting cavity for larger holes that might result from the deletion of walkers which can lead to a bad estimate of the cavity's surface area and volume and maybe also have an impact on the quality of the polarization potential.

Tag: isc_kill_ratio (control.in)

Usage: isc_kill_ratio fraction

Purpose: This keyword can be helpful when the walker dynamics run does not converge due to trapped walkers by killing the worst fraction of walkers at each neighbor list update step (also see isc_update_nlist_interval).

fraction is a non-negative real number much smaller than 1. Default: 0.0

As the number of possibly trapped walkers depends a lot on the shape of the electron density, it is rather difficult to give a recommendation about a sensible value for fraction. In case walkers get stuck, we propose to use a rather conservative kill ratio of 1×10^{-3} and only increase it if necessary.

Tag: isc_update_nlist_interval (control.in)

Usage: isc_update_nlist_interval num

Purpose: This keyword triggers a re-evaluation of the neighbor lists in the density walkers dynamics simulation after every num steps.

num is a positive integer number. Default: 50

Tag: isc_dynamics_friction (control.in)

Usage: isc_dynamics_friction fric

Purpose: The value of fric determines how much "kinetic energy" is removed from the walkers in every step of the simulations via a simple velocity scaling.

fric is a real number between 0 and 1. Default: 0.1

A value of 0 for fric means that no energy is removed from the system which may lead to a bad convergence behavior. On the other hand, a value of 1 means that all kinetic energy is removed at each step which tends to slow down the rate of convergence.

Tag: `isc_dt` (control.in)

Usage: `isc_dt delta`

Purpose: Determines the "time" step of the walker dynamics simulation.

`delta` is a positive real number. Default: `0.1`

Note: Since this is no actual physical quantity, arbitrary time units are used.

Tag: `isc_rho_k` (control.in)

Usage: `isc_rho_k k`

Purpose: Determines the force constant `k` for the "density" force, i.e. the harmonic force that pulls the walkers along the density gradient to the specified iso-density value.

`k` is a positive real number. Default: `1.0`

Note: Since this is no actual physical quantity, arbitrary time units are used.

Tag: `isc_rep_k` (control.in)

Usage: `isc_rep_k k`

Purpose: Determines the force constant `k` for the repulsive interaction between walkers perpendicular to the density gradient.

`k` is a positive real number. Default: `0.01`

Note: Since this is no actual physical quantity, arbitrary time units are used.

Tag: `isc_g_k` (control.in)

Usage: `isc_g_k k`

Purpose: Determines the force constant `k` for the "gravitational" force that drags walkers to the center of gravity of the solute in case the local density gradient is too small.

`k` is a positive real number. Default: `2.0`

Usually this should not happen, but when walkers move too far away from the solute, the density gradient becomes very small and its direction is unreliable due to numerical noise (see `isc_gradient_threshold`). In this case, the walker is dragged to the center of the solute until the density gradient is again large enough. Note: Since this is no actual physical quantity, arbitrary time units are used.

Tag: `isc_gradient_threshold` (control.in)

Usage: isc_gradient_threshold thsq

Purpose: When the squared norm of the electron density gradient at the position of a walker is less than thsq, this gradient is considered unreliable. Instead, a simple "gravitational" force towards the center of the solute is applied.

thsq is a positive real number. Default: 1×10^{-8}

The force constant of the "gravitational" force is determined by isc_g_k .

debug

Tag: mpe_timing_output_level (control.in)

Usage: mpe_timing_output_level int

Purpose: Defines how deep timing analysis of the MPE method will regress into subroutines. Default: 2

Tag: isc_cavity_restart (control.in)

Usage: isc_cavity_restart filename

Purpose: Read the solvation cavity from restart file (if available) and write new cavity to same file.

filename is the name of the restart file.

Specifying this keyword is almost equivalent to specifying both isc_cavity_restart_read and isc_cavity_restart_write with the same filename option except that with isc_cavity_restart the program does not abort when there is no restart file to read from.
Note: This keyword is intended for debugging purposes. Do not rely on the current structure of the cavity restart file as it might change in the future.

Tag: isc_cavity_restart_read (control.in)

Usage: isc_cavity_restart_read filename

Purpose: Read the solvation cavity from restart file instead of constructing a new one.

filename is the name of the file (in .xyz format) containing the cavity points and normal vectors. Additionally, a file <filename>.bin is written which contains the entire cavity information in not human-readable form. While the primary purpose of the former is visualization, the latter is the actual restart file.

Note: This keyword is intended for debugging purposes. Do not rely on the current structure of the cavity restart file as it might change in the future.

Tag: `isc_cavity_restart_write` (control.in)

Usage: `isc_cavity_restart_write filename`

Purpose: Write the cavity to the specified restart file once created.

`filename` is the name of the restart file (in `.xyz` format). If additionally `<filename>.bin` is present, the cavity is read from the latter instead. Note that the `.xyz` file has to be present in both cases. While this file itself is sufficient to create a cavity, only the `.bin` file allows for a fully deterministic restart.

Note: This keyword is intended for debugging purposes. Do not rely on the current structure of the cavity restart file as it might change in the future.

Tag: `isc_record_cavity_creation` (control.in)

Usage: `isc_record_cavity_creation filename num`

Purpose: Controls the output of snapshots during the cavity generation process. When `num` is positive, every `num` steps an XYZ snapshot of the cavity is written to file `filename`. For other choices of `num`, no output will be generated.

`filename` is of type string.
`num` is of type integer. Default: `0`

This keyword is intended for debugging purposes. Note that the size of the output file can become very large!

deprecated

Tag: `mpe_n_centers_ep` (control.in)

Usage: `mpe_n_centers_ep n`

Purpose: Defines the number of centers used for the expansion of the polarization potential outside of the cavity.

`n` is a positive integer number. Default: number of centers for the Hartree potential expansion (cf. 3.7)

The first `n` centers defined in `geometry.in` are used as expansion centers. The default is to use all of them. Only change this value if you fully understand what you are doing and why you want to do this!

Tag: `mpe_n_boundary_conditions` (control.in)

Usage: `mpe_n_boundary_conditions nbc`

Purpose: Determines the number of boundary conditions imposed at every point on the cavity interface.

Valid choices for `nbc` are 2 and 4. Default: `2`

As outlined in Ref. [215], there are at least two more boundary conditions other than Eqns. 3.40a and 3.40b that can be imposed on the electrostatic potential / field / flux at a dielectric interface. The default is to enforce continuity of the potential and continuity of the dielectric flux perpendicular to the interface, i.e. `nbc` equals 2. Furthermore, continuity of the electric field parallel to the interface can be imposed, i.e. `nbc` equals 4. However, this should automatically be satisfied by the former two boundary conditions and—in the best case—only leads to a higher order correction of the fit. *Warning:* The non-default has not been tested thoroughly. Verify your results carefully when using it!

Tag: `isc_calculate_surface_and_volume` (control.in)

Usage: `isc_calculate_surface_and_volume bool`

Purpose: Determines whether the surface area and volume of the cavity are calculated.

`bool` is of Boolean type. Default: `.true.`

As the only currently implemented `mpe_nonelectrostatic_model` `linear_OV` requires the calculated measures, this flag is automatically turned on when it has been turned off but is needed.

Tag: `mpe_xml_logging` (control.in)

Usage: `mpe_xml_logging filename level`

Purpose: Controls the MPE module's internal XML logging output.

`filename` specifies the name of the log file to be written. Default: `mpe_interface.xml`
`level` defines the detail of the output. Supported log levels are: `off`, `basic`, `medium`, and `detailed`. Default: `off`

This keyword is intended for debugging purposes. Note: Depending on the log level, the size of the output can become quite large.

3.17.2 SMPB Implicit Electrolyte Model

In FHI-aims, implicit solvation effects or electrolyte effects (z:z electrolytes) can be included by solving the Stern- and finite ion-size *Modified Poisson-Boltzmann* equation (*(S)MPBE*) in each SCF step:

$$\nabla \cdot [\varepsilon[n_{\mathrm{el}}(\boldsymbol{r})]\nabla v(\boldsymbol{r})] \;=\; -4\pi n_{\mathrm{sol}}(\boldsymbol{r}) - 4\pi n_{\mathrm{ion}}^{\mathrm{MPB}}(\boldsymbol{r}) \quad , \tag{3.41}$$

with

$$n_{\mathrm{ion}}^{\mathrm{MPB}}(\boldsymbol{r}) \;=\; z\left[c_{+}^{\mathrm{s}}(\boldsymbol{r}) - c_{-}^{\mathrm{s}}(\boldsymbol{r})\right] \quad , \tag{3.42}$$

where $\varepsilon[n_{\mathrm{el}}]$ is a parameterized function of the electron density, v is the electrostatic potential, n_{sol} is the solute charge density consisting of electrons and nuclei and $n_{\mathrm{ion}}^{\mathrm{MPB}}$ is the ionic charge density modeled as a function of the exclusion function $\alpha_{\mathrm{ion}}[n_{\mathrm{el}}]$ being parameterized via the electron density and the electrostatic potential v. The implementation so far supports different kind of models for the ionic charge density, that is the modified, the linearized or the standard PBE. All models include a model for the Stern layer by a repulsion of the ions from the solute modeled via $\alpha_{\mathrm{ion}}[n_{\mathrm{el}}]$ and the size-modified version also a finite ion size a. Parameterizations are needed for the dielectric function (n_{min} and n_{max}) and nonmean-field interaction of solvent with solute ($(\alpha+\gamma)$ and β) which are readily available for water solvents but have to be obtained first for other solvents. Ionic parameters (ion size a and Stern layer defining parameters $d_{\alpha_{\mathrm{ion}}}$ and $\xi_{\alpha_{\mathrm{ion}}}$) are not known so far and we are currently working on deriving them.

The energies are outputted in the end of FHI-aims under the header `MPBE Solvation Additional Free Energies`:

- `Total energy` = Electrostatic part of the energy. This does NOT consider yet any non-electrostatic corrections (see next term)
- `Free Energy in Electrolyte` = $\Omega_{\circ}$ in ref. [201]. Free energy of solute in electrolytic environment, which is `Total energy` + `Nonelectrostatic Free Energy` + `Additional Nonelstatic MPBE Solvation Energy`, where `Total energy` is the normally outputted energy in Aims (electrostatic part)
- `Surface Area of Cavity` = quantum surface of solvation cavity
- `Volume of Cavity` = quantum volume of solvation cavity
- `Nonelectrostatic Free Energy` = non-electrostatic part of solvation energy due to solute-solvent interactions, $\Omega^{\mathrm{non-mf}}$ in the publication
- `Additional Nonelstatic MPBE Solvation Energy` = non-electrostatic part of free energy due to ions. For ion-free calculations this is zero.

For more details see [201, 202]. If you want to do any calculations considering solvent or ion effects, please contact the authors, we are happy to help and cooperate.

The keywords listed here are the main part of all keywords. Some of the keywords were left out because they are highly experimental, if one is interested in more options, please contact the authors.

Tags for general subsection of `control.in`:

Tag: `solvent mpb` (control.in)

Usage: `solvent mpb`

Purpose: Switches MPB solvent effects on.

Restriction: Only for cluster systems (no periodic systems).

`solvent mpb` sub-tag: `dielec_func` (control.in)

Usage: `dielec_func type parameters`

Purpose: Define the dielectric function.

`type` integer describes the type of dielectric function used, `type`=0 Fattebert & Gygi[68] or `type`=1 Andreussi & Marzari[6]

`parameters` settings for dielectric function, separated by space:
`type`=0: bulk dielectric constant $\epsilon^{\mathrm{s,bulk}}$, β, n_0

`type`=1: bulk dielectric constant $\epsilon^{\mathrm{s,bulk}}$, n_{min}, n_{max}

Default: `1 78.36 0.0001 0.005`[6]

`solvent mpb` sub-tag: `ions_{parameter}` (control.in)

Usage: `ions_{parameter} parameter`

Purpose: Set the parameters defining the ions in the electrolyte. In our recent publication[202] we explain how to choose these for different monovalent salt solutions.

`parameter` {parameter} =

- *temp* (temperature (K))
- *conc* (bulk concentration $c^{\mathrm{s,bulk}}$ (mol/L))
- *charge* (z)
- *size* (lenght of lattice cell *a* (Å))
- *kind* (0 for sharp step function, 1 for smooth function)
- *mod_alpha* ($d_{\alpha_{\mathrm{ion}}}$,$\xi_{\alpha_{\mathrm{ion}}}$)

Defaults: T = 300K, $c^{\mathrm{s,bulk}}$ =1M,z=1,a=5, `kind` = 1, $d_{\alpha_{\mathrm{ion}}}$ =`0.5`,$\xi_{\alpha_{\mathrm{ion}}}$ =`1.0`

Remarks: The inclusion of a second α_{ion} function for the anions is experimental and should not be used. The use of a sharp cutoff function for α_{ion} is not recommended, not properly implemented and just there for testing purposes.

---

solvent mpb sub-tag: SPE_{setting} (control.in)

Usage: `SPE_{setting} parameter`

Purpose: Change numerical parameters of the SPE solver.

`parameter` {setting} =

- *lmax* (maximum angular momentum $l_{\max}$ of multipole expansion and of all species)
- *conv* (τ_{MERM}, η, separated by space)
- *cut_and_lmax_ff* (distance from atom centers at which far field is turned on – `multipole_radius_SPE`, $l_{\max}^{\mathrm{ff}}$ – maximum angular momentum in the far field, separated by space)

Defaults: $l_{\max}$ = `max(l_hartree)`, τ_{MERM} = `1e-10`, η =`0.5`, $l_{\max}^{\mathrm{ff}} = l_{\max}$, `multipole_radius_SPE` is per default not used and the species dependent `multipole_radius_free` + 2.0 is used as far field cutoff radius.

Remarks: Due to our present tests, we do not recommend to use $l_{\max}^{\mathrm{ff}} < l_{\max}$, the errors in the energies at the normal cutoff radius are too big. τ_{MERM} =`1e-8` can be enough in most cases and speed up the calculation. The species dependend `l_hartree` can be by implementation not larger than $l_{\max}$, so it is reduced to $l_{\max}$ if higher for the SPE solver.

---

solvent mpb sub-tag: dynamic_{quantity}_off (control.in)

Usage: `dynamic_{quantity}_off`

Purpose: If these keywords are used, {quantity} is parameterized before the SCF cycle from the superposition of free energy densities.

{quantity} =

- *cavity* dielectric function ε
- *ions* exclusion function α_{ion}

Default: both keywords not used by default, so both quantities are calculated self-consistently by parameterizing it with the full electron density.

---

solvent mpb sub-tag: delta_rho_in_merm (control.in)

Usage: delta_rho_in_merm

Purpose: Setting this keyword, evaluates the change of the source term $q - \frac{1}{4\pi}\hat{L}_1\delta v_{n+1}$ during the MERM iteration and solves the SPE for this change rather than the full source density.

Default: Not used. This keyword is under development and experimental, do not use it, yet.

---

solvent mpb sub-tag: nonsc_Gnonmf (control.in)

Usage: nonsc_Gnonmf

Purpose: Setting this keyword, calculates the free energy term $\Omega^{\mathrm{non-mf}}$ as a post-correction after the convergence of the SCF cycle, so no Kohn-Sham correction is added which would normally arise from this term. This has been proven to give very similar results for solvation energies like the fully self-consistent calculation of this term. Since people observed numerical instabilities due to this term, sometimes it might be better to set this flag.

Default: Not used. Fully self-consistent evaluation of $\Omega^{\mathrm{non-mf}}$

---

solvent mpb sub-tag: Gnonmf_FD_delta (control.in)

Usage: Gnonmf_FD_delta parameter

parameter Δ parameter defining the thickness of the cavity

Purpose: Used to calculated the quantum surface S and volume V to evaluate the free energy contribution $\Omega^{\mathrm{non-mf}}$

Default: 1e-8

---

solvent mpb sub-tag: not_converge_rho_mpb (control.in)

Usage: not_converge_rho_mpb

Purpose: Setting this keyword, runs a vacuum calculation first and then subsequently solves the MPBE once with the vacuum electron density and then outputs all energetics.

Default: Not used. This could be of interest for either very big systems to get first approximations without running the Newton method in each SCF step but only once, but of course then does not involve any self-consistent solution of the coupled Kohn-Sham and MPB equations. Originally, this feature was introduced to evaluate electrostatic potentials and compare them to other codes, like e.g. FEM codes.

---

solvent mpb sub-tag: solve_lpbe_only (control.in)

Usage: `solve_lpbe_only logical`

Purpose: Instead of solving the MPBE, solve the linearized version of this, also called the LPBE. For neutral molecules electrostatic fields are often small, so the LPBE electrostatic potential is often a good approximation to the true MPBE potential. The solution of the LPBE can be done directly using the MERM without the Newton method and is therefore faster for most cases.

`logical` if `.True.`, use the LPB electrostatic potential, but the MPB free energy expression which contains additional entropic terms compared to the LPB expression.

Default: By default the MPBE is solved, so this is not used.

`solvent mpb` sub-tag: `MERM_in_SPE_solver` (control.in)

Usage: `MERM_in_SPE_solver logical`

Purpose: Do the MERM iterations inside the `SPE_solver.f90` routine without updating δv_{n+1} on the full integration grid at each step, but only on the points where we actually need it to form the source term. By this, we can gain speed, especially for $c^{\mathrm{s,bulk}} = 0$.

`logical`

Default: `.true.` Remark: In general the both options should give exactly the same result at convergence. If any difficulties arise, one is however recommended to try the `.False.` options, since it should be the more stable version of the solver.

`solvent mpb` sub-tag: `MERM_atom_wise` (control.in)

Usage: `MERM_atom_wise logical`

Purpose: Do the MERM iterations for each atom separately, i.e. we write eq. (33)[201] (Generalized Poisson or LPB-kind of equation) as:

$$\left(\nabla\left[\varepsilon\nabla\right] - h^2[v_n]\right)\delta v_{n+1,\mathrm{at}} = -4\pi\varepsilon p_{\mathrm{at}} q[v_n] \tag{3.43}$$

$$\delta v_{n+1} = \sum_{\mathrm{at}} \delta v_{n+1,\mathrm{at}} \tag{3.44}$$

$$q[v_n] = \sum_{\mathrm{at}} p_{\mathrm{at}} q[v_n] \tag{3.45}$$

In order to perform the MERM iterations for each atom, the full grid of the respective atom has to be used, i.e. also the electron density needs to be updated on points where commonly the `partition_tab` is vanishing. However, by this we avoid the cross-update of atomic potentials on the atomic grid of other atoms as needed in the original method and this is usually most costly in particular for larger systems. In terms of convergence with the maximum angular momentum l_{max}, this method performs a bit worse than the original method, which is why we recommend to use $l_{\mathrm{max}} = 8$ for production runs. Still this method should be faster also with this higher accuracy in the multipole expansion.

`logical`

Default: `.false.`

Remark: Using this flag will automatically set `MERM_in_SPE_solver = .True.`.

`solvent mpb` sub-tag: `set_nonelstat_params` (control.in)

Usage: `set_nonelstat_params value value`

Purpose: Set the parameters for the nonelectrostatic solvent-solute interactions.

`value value` two real numbers, $\alpha + \gamma$ (dyn/cm) and β (GPa), separated by space.

Default: $\alpha + \gamma = 50$ dyn/cm, $\beta = -0.35$ GPa

3.18 Hubbard corrected DFT (DFT+U)

Standard semi-local DFT functionals like LDA or GGA suffer from improper self-interaction error (SIE) cancellation. As a results this functional utterly fail when it comes to the description of systems which are characterized by localized electron states. One specific approach cure for this drawback is to use *hubbard corrected* DFT also known as DFT+U or LDA+U. In this approach one adds a correction to the LDA or GGA Hamiltonian which is inspired by the Hubbard Model [112]. The correction allows to reduce the self-interaction error in systems, which are characterized by correlated states, significantly [8]. Its great strength lies in the simplicity of its corrective term and in the fact that its computational cost is only marginally higher compared to LDA or GGA. Thus, the ability to localize electrons and its computational efficiency make DFT+U to a suitable tool for studying systems in PBC supercell calculations [108].

In the following some of the main features of DFT+U, which are specific to the implementation in FHI-aims, are addressed.

Incorporation of the Hubbard model into the normal approximate DFT description leads to the following DFT+U energy functional:

$$E_{\mathrm{DFT+U}}\left[\rho\left(\mathbf{r}\right)\right] = E_{\mathrm{DFT}}\left[\rho\left(\mathbf{r}\right)\right] + E_{\mathrm{U}}^{0}\left[n_{Im}\right] - E_{\mathrm{dc}}\left[n_{Im}\right]. \tag{3.46}$$

Here, E_{DFT} is the standard DFT energy functional on a LDA or GGA level of theory. E_{U}^{0} depends on the orbital occupancy n_{Im} of the correlated states at site I and represents the energy correction according to the Hubbard Hamiltonian. However, by simply adding E_{U}^{0} to E_{DFT}, one runs into a double-counting issue of the coulomb interaction, because all the electron-electron interactions are already taken into account in LDA or GGA. Furthermore, the DFT Hamiltonian explicitly depends on the charge density, while the Hubbard Hamiltonian is written in the orbital representation. Therefore, one can not build a direct link between both descriptions and a simple subtraction of the double-counting is not possible. As a consequence, the dc functional E_{dc} is not uniquely defined and different formulations of E_{dc} can lead to different results of the calculation [241]. Within FHI-aims we offer three different double-counting correction strategies (see `plus_u_petukhov_mixing`), the fully-localized limit (FLL), the around mean field (AMF) approximation and a interpolation scheme where the double counting correction is calculated in a self-consistent manner [189]. We strongly recommend to choose the FLL as double-counting correction, as it is the most common one used in literature.

The last two terms on the r.h.s. of eq. 3.46 are usually combined to one energy correction, E_U. One arrives at following expression,

$$E_{\mathrm{DFT+U}}\left[\rho\left(\mathbf{r}\right)\right] = E_{\mathrm{DFT}}\left[\rho\left(\mathbf{r}\right)\right] + E_{\mathrm{U}}\left[n_{Im}\right]. \tag{3.47}$$

As briefly mentioned, the orbital occupancies n_{Im} are the occupation numbers of localized orbitals, where m is the state index which usually runs over the eigenstates of L_z for a certain angular momentum l. With other words, n_{Im} are the occupation numbers of a specific shell of orbitals, located at a certain atom. The definition of a shell is best explained by using an example. If a DFT+U treatment is requested for the $3d$ electrons of a single first row transition metal, then a shell represents the five $3d$-orbitals for each spin type.

3.18.1 DFT+U correction as it is implemented in FHI-aims

So far this was just a brief sketch of the DFT+U approach in general. In the following we present the precise definition of DFT+U how it is implemented in FHI-aims. Without loss of generality we only show the equations with FLL as double-counting correction.

$$\begin{aligned} E_{\mathrm{U}}^{\mathrm{FLL}}\left[\{n_{Imm'}^{\sigma}\}\right] &= E_{\mathrm{U}}^{0}\left[\{n_{Imm'}^{\sigma}\}\right] - E_{\mathrm{dc}}\left[\{n_{Imm'}^{\sigma}\}\right] \\ &= \frac{1}{2}\sum_{\sigma,I} U_{\mathrm{eff}}^{I} Tr\left[\mathbf{n}_{I}^{\sigma}\left(1-\mathbf{n}_{I}^{\sigma}\right)\right] \\ &= \frac{1}{2}\sum_{\sigma,I} U_{\mathrm{eff}}^{I}\left[Tr\left(\mathbf{n}_{I}^{\sigma}\right) - Tr\left(\mathbf{n}_{I}^{\sigma}\mathbf{n}_{I}^{\sigma}\right)\right]. \end{aligned} \tag{3.48}$$

These functional is known as the spherically averaged form of DFT+U. It was first proposed by Dudarev *et al.*[61] and it is also rotational invariant. In this formulation, the effective on-site interactions enter via their spherical atomic averages. This is justified by the fact, that localized states still have atomic character and hence, spherical symmetry. In fact, for most materials this definition gives good results.

It should be pointed out, that U_{eff} can be seen as an effective value of the coulomb interaction that also includes exchange corrections. This parameter has to be specified by hand, so far, no possibility is implemented to calculate this parameter self-consistently. Common to all approaches is that all the calculated results sensitively depend on the applied U_{eff} value. This value not only depends on the atom for which DFT+U is applied. It also depends on the surroundings of the atom, the lattice parameters and physical conditions. Furthermore, it also depends on the localized basis set of the underlying quantum DFT code. This limits the comparability of different values in a strong way. In general, for each DFT+U implementation and system, one should recalculate U_{eff}.

The most important quantity in equation 3.48 is the so called DFT+U occupation matrix $\mathbf{n}$. This matrix simply tells us how many electrons are in a certain shell on a certain atom. The problem here is the inability to break down the total charge density into atom specific contributions. Or in other words, there is no proper operator for counting the number of electrons on an atom. Hence, the choice of the occupation matrix will affect the outcome of a calculation. Within FHI-aims we offer two specific choices: the on-site representation of the occupation matrix

$$n_{Imm'}^{\sigma}(\mathrm{on-site}) = D_{Im,Im'}^{\sigma} \tag{3.49}$$

and

$$n_{Imm'}^{\sigma}(\mathrm{dual}) = \frac{1}{2}\sum_{Jk}\left[D_{Im,Jk}^{\sigma} S_{Jk,Im'} + S_{Im,Jk} D_{Jk,Im'}^{\sigma}\right]. \tag{3.50}$$

The latter is known as the dual representation [94]. Within the dual representation the occupation numbers are calculated in a similar way as in the Mulliken analysis. The main difference between both is that the on-site version only accounts for overlaps within a specific sub shell on an certain atom. The dual representation also accounts for the overlap with the surrounding atoms. It is emphasized that all general aspects of DFT+U are met by all matrix representations. Furthermore, more detailed studies regarding the

performance of the occupation matrix for various transition metal oxides revealed that in principle there is no definition which is clearly the best [220]. Unfortunately, we only offer forces for the on-site representation. The on-site version is also the default occupation matrix in FHI-aims and we strongly recommend to use it.

By now, one might have noticed that DFT+U is by far not a black box method and it gets even worse if one considers in detail how the occupation matrix is constructed. In general, each occupation matrix can be expressed in terms of a local projector operator, $\hat{P}^{\sigma}_{Imm'}$. The (m,m')-th element of a occupation matrix at site I is then given by

$$n^{\sigma}_{Imm'} = \sum_{\gamma} f_{\gamma} \langle \Psi^{\sigma}_{\gamma} | \hat{P}^{\sigma}_{Imm'} | \Psi^{\sigma}_{\gamma} \rangle \,. \tag{3.51}$$

For example for the on-site projection operator this would lead to

$$\hat{P}^{\sigma}_{Imm'}(on\text{-}site) = |\tilde{\phi}^{\sigma}_{Im'}\rangle \langle \tilde{\phi}^{\sigma}_{Im}| \,. \tag{3.52}$$

Here, $\tilde{\phi}_{Im}$ denotes the dual basis functions which are defined in terms of the inverse overlap matrix $\mathbf{S}^{-1}$,

$$|\tilde{\phi}^{\sigma}_{m}\rangle = \sum_{I'm'} S^{-1}_{Im,I'm'} \, |\tilde{\phi}^{\sigma}_{m'}\rangle \,. \tag{3.53}$$

The question now is which basis functions should be used in the projection? As default we are using the *atomic* type basis functions of the minimal basis set in FHI-aims. Here we automatically assume, as they are *atomic* like basis functions, that they will contribute most to the localized states. However, in general it is not known if other basis functions should also be included in the DFT+U projection e.g. *tier1* $3d$ if one deals with first row transition metals. Usually one can notice that by a strange behavior of the occupation matrix during the scf-cycle (occupation numbers drop to zero as the electrons occupy other basis functions). For that purpose we offer to include also other basis functions in the description of DFT+U (see `plus_u_use_hydors`). We also like to highlight the corresponding paper related to our implementation where we address fundamental issues of DFT+U in a LCAO electronic structure code. However, do not panic, for most of the systems the default settings should be sufficient enough.

So far, we presented DFT+U in quite some detail. However, we just wanted to highlight that DFT+U is far from being a black box method. However, the handling of a actual DFT+U calculation in FHI-aims is quite easy. One just have to specify the double-counting correction first via the `plus_u_petukhov_mixing` . Afterwards one can specify the U value and the angular momentum shell to which DFT+U should be applied for each species. Of course one can specify different U values for different species in a simulation. Only for hard cases where convergence can not be reached easily, it is quite useful to checkout the other keywords. Some of them can be quite useful such as the `plus_u_matrix_control` . Here, one first converges the density with help of a fixed occupation matrix which can be edited by hand. Afterwards one can use the restart information to calculate everything self-consistently. This can be quite useful as it turns out that DFT+U is quite sensitive to the initial guess of a calculation. Furthermore, it is quite useful also to start from a LDA or GGA ground state density.

Tag: `plus_u_petukhov_mixing` (control.in)

Usage: `plus_u_petukhov_mixing mixing_factor`

Purpose: *only for DFT+U.* Allows to fix the mixing factor between AMF and FLL contribution of the double counting correction [189].

`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. [189].
We strongly recommend to use the FLL.

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 with in the shell are either fully occupied or empty. The self consistent mixing of both limits improves the handling of the intermediate range (see Ref. [189]).

Tag: `plus_u_use_mulliken` (control.in)

Usage: `plus_u_use_mulliken`

Purpose: *only for DFT+U.* Allows to switch from on-site representation to the dual representation of the occupation matrix.

Default is the on-site representation. Forces are not provided for the dual representation.

Tag: `plus_u_out_eigenvalues` (control.in)

Usage: `plus_u_out_eigenvalues`

Purpose: *only for DFT+U.* Allows to calculate the eigenvalues of the self-consistent DFT+U occupation matrix at the end of a run.

Tag: `plus_u_matrix_control` (control.in)

Usage: `plus_u_matrix_control`

Purpose: *only for DFT+U.* Allows to write the self-consistent occupation matrix to a file `occupation_matrix_control.txt`. If the file is already present in the calculation folder, the occupation matrix is not calculated during the run. It will be read out from that file instead. The occupation matrix is then fixed during the complete run.

This is extremely useful because one can simply edit the file and manipulate the matrix according to some specific spin configuration. Consider to use it with restart options.

Tag: `plus_u_matrix_release` (control.in)

Usage: `plus_u_matrix_release convergence_accuracy`

Purpose: *only for DFT+U.* If this keyword is present in combination with `plus_u_matrix_control` the occupation matrix is first fixed to the matrix from the `occupation_matrix_control.txt` file until some certain convergence criteria of the total energy is fulfilled. Afterwards the occupation matrix is calculated self-consistently again.

`convergence_accuracy` this threshold specifies the convergence in total energy from which point on the occupation matrix should be calculated self-consistently. The value is a floating point number.

Tag: `plus_u_use_hydros` (control.in)

Usage: `plus_u_use_hydros`

Purpose: *experimental — only for DFT+U.* If this keyword is present also hydrogen like basis functions are included in the DFT+U correction.

The code builds up a simple linear combination of all basis functions which contribute to the angular momentum channel to which DFT+U is applied. All basis functions will contribute equally (see also `hubbard_coefficient`).

Tag: `plus_u_matrix_error` (control.in)

Usage: `plus_u_matrix_error`

Purpose: *experimental — only for DFT+U.* Calculates the idempotence error of the occupation matrix $\mathrm{Tr}\left(\mathbf{n}-\mathbf{nn}\right)$

Tag: `plus_u_ramping_accuracy` (control.in)

Usage: `plus_u_ramping_accuracy convergence_accuracy`

Purpose: *experimental — only for DFT+U.* If this keyword is present the calculation starts at U = 0 eV. If the specified convergence accuracy of the total energy is reached, the U value is slightly increased. This is will be done until the final U value is reached.

`convergence_accuracy` Floating point number. Defines the convergence accuracy from which on the U value is increased stepwise by a certain increment. The increment can be specified with the `plus_u_ramping_increment` keyword.

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

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

Usage: `plus_u n l U`

Purpose: *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.

The U here defined equals U_{eff} in eq. 3.48.

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

Usage: `hubbard_coefficient` c_1 c_2 c_3 c_4

Purpose: *experimental — only for DFT+U.* Only works in combination with the `plus_u_use_hydros` keyword. Allows the user to specify his one projector function for DFT+U as long as this function can be represented by basis functions contributing to a specific angular momentum which is given by the `plus_u` keyword. Only four basis functions are allowed in the expansion and the order corresponds to their appearance in the control in.

c_1 expansion coefficient of the first basis function
c_2 expansion coefficient of the second basis function
c_3 expansion coefficient of the third basis function
c_4 expansion coefficient of the 4th basis function

If a basis function should not be part of the linear combination the corresponding coefficient should be set to 0. Keep in mind that aims performs an on-site orthogonalization of all basis function located at a certain atom. This means that the radial shape of a basis function might be different from that, one would expect from the `control.in` definition. Within DFT+U all basis functions are orthogonalized w.r.t. the *atomic* basis functions.

---

species sub-tag: plus_u_ramping_increment (control.in)

Usage: plus_u_ramping_increment increment

Purpose: *experimental — only for DFT+U.* Specifies the the step by which the U value should be increased. Works only in combination with plus_u_ramping_accuracy

increment specified in eV.

3.19 C_6/R^6 corrections for long-range van der Waals 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. [224]. 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_convergence_threshold` (control.in)

Usage: `vdw_convergence_threshold value`

Purpose: When using the vdW correction based on Hirshfeld partitioning of the electron density (as described in Tkatchenko and Scheffler 2009, Ref. [224]) in a periodic system, this sets the energy convergence threshold for the supercell sum over the TS components.
`value`: A small positive number (in eV). Default: For unit cells with less than 100 atoms: 10^{-6} eV. For structures with unit cell sizes above 100 atoms, the default is adjusted to $n_{\text{atoms}} \cdot 10^{-8}$ eV.

Note that the vdw part of the forces may be separately converged to (if set) `sc_accuracy_forces`.

Tag: `vdw_correction_hirshfeld` (control.in)

Usage: vdw_correction_hirshfeld

Purpose: Enables the vdW correction based on Hirshfeld partitioning of the electron density (as described in Tkatchenko and Scheffler 2009, Ref. [224]). 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 or vdw_convergence_threshold and the forces (if requested) are converged to within sc_accuracy_forces .
No other input required.

This method is commonly referred to as the Tkatchenko-Scheffler method. The procedure is as follows. *First*, the normal self-consistency cycle is completed for a semilocal or hybrid density functional, most commonly PBE or PBE0. *Second*, the resulting self-consistent electron density is used to create interatomic (pairwise) C_6 coefficients. A simple pairwise van der Waals term is then added once to the self-consistent total energy from the preceding semilocal or hybrid functional. In other words, the Tkatchenko-Scheffler method is normally employed as a post-processing term in a non-self-consistent way, not during the self-consistency cycle. Since it needs to be combined with a different density functional, you would normally use it like this (example for "PBE+vdW"):

```
xc pbe
vdw_correction_hirshfeld
```

Three more caveats: (1) Do not use this method together with the local-density approximation (LDA) unless you know exactly what you are doing. The LDA already contains a spurious interaction term that will lead to very strange results if added to a pairwise van der Waals term. (2) Do not simply apply this method to a metallic system unless you know what you are doing. (3) This is also not the (very different) functional commonly known as the Langreth-Lundqvist or vdw-DF functional. [59] FHI-aims contains at least two implementations of vdw-DF for those who are interested, but either implementation is much slower than the Tkatchenko-Scheffler pairwise interatomic sum.

Tag: `vdw_correction_hirshfeld_sc` (control.in)

Usage: vdw_correction_hirshfeld_sc

Purpose: Enables the self-consistent version of the vdW correction based on Hirshfeld partitioning of the electron density (see Tkatchenko and Scheffler 2009, Ref. [224]). In a periodic calculation, the energy is converged with the same criteria of the *a posteriori* approach: vdw_correction_hirshfeld .

This flag adds the Tkatchenko-Scheffler vdW functional as a part of the given exchange-correlation (XC) functional. In a self-consistent scheme, the contribution of the vdW potential, $v_{\mathrm{vdW}}[n(\mathbf{r})] = \delta E_{\mathrm{vdW}}[n(\mathbf{r})]/\delta n(\mathbf{r})$, is added to the XC potential to form the total effective potential in the Kohn-Sham equations. As a result, the van der Waals interatomic contributions affect the total electron density and are computed at each self-consistent cycle, until convergence is reached. In this way, it is possible to evaluate the effects of vdW interactions on the electron density and electronic properties, going beyond the vdW *a posteriori* correction of the total DFT energy.

Note: Do not use this self-consistent flag during a relaxation. The self-consistent forces are not implemented (yet) for the Tkatchenko-Scheffler vdW functional and this will lead to inconsistency errors.

Tag: `vdw_ts` (control.in)

Usage: `vdw_ts [option=value...]`

Alternative implementation of the TS method via the Libmbd library [103] that provides some extra features compared to the `vdw_correction_hirshfeld` and `vdw_correction_hirshfeld_sc` implementations. The main difference is the use of the Ewald summation to sum the $1/R^6$ sum in the periodic case, which makes the use of the `vdw_convergence_threshold` keyword obsolete. Furthermore, forces and stress are available also in the self-consistent case. This implementation does not honor the `vdw_pair_ignore` and `hirshfeld_param` keywords.

- `self_consistent=<logical>` [default: `.false.`] turns on self-consistency as explained under `vdw_correction_hirshfeld_sc` .
- `vdw_params_kind=<string>` [default: `"ts"`] specifies the set of free-atom vdW parameters (α_0, C_6, R_{vdW}) used. `"ts"` uses the original set of parameters, `"tssurf"` uses values from the so-called vdW$^{\text{surf}}$ approach [206] for some of the elements. The used parameters are listed here.

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 atom$_i$ atom$_j$ C_{6ij} R^0_{ij} d", where C_{6ij} is the C_6 coefficient for the interaction between atom$_i$ and atom$_j$, R^0_{ij} 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 C6-correction, eg. such that metallic slabs are not affected internally by introducing C6-interactions. Does not apply to the `vdw_ts` keyword.

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. [224]
Purpose: To explicitly allow setting the parameters for the Tkatchenko-Scheffler van der Waals correction. Does not apply to the `vdw_ts` keyword.

3.20 Many-Body Dispersion (MBD) method

The many-body dispersion (MBD) method calculates the long-range van der Waals (vdW) energy of a system by modeling the response of atoms with quantum harmonic oscillators coupled via a dipole potential [226, 225]. Two versions of MBD are available in FHI-aims, MBD@rsSCS and MBD-NL, both via the included Libmbd library [103].

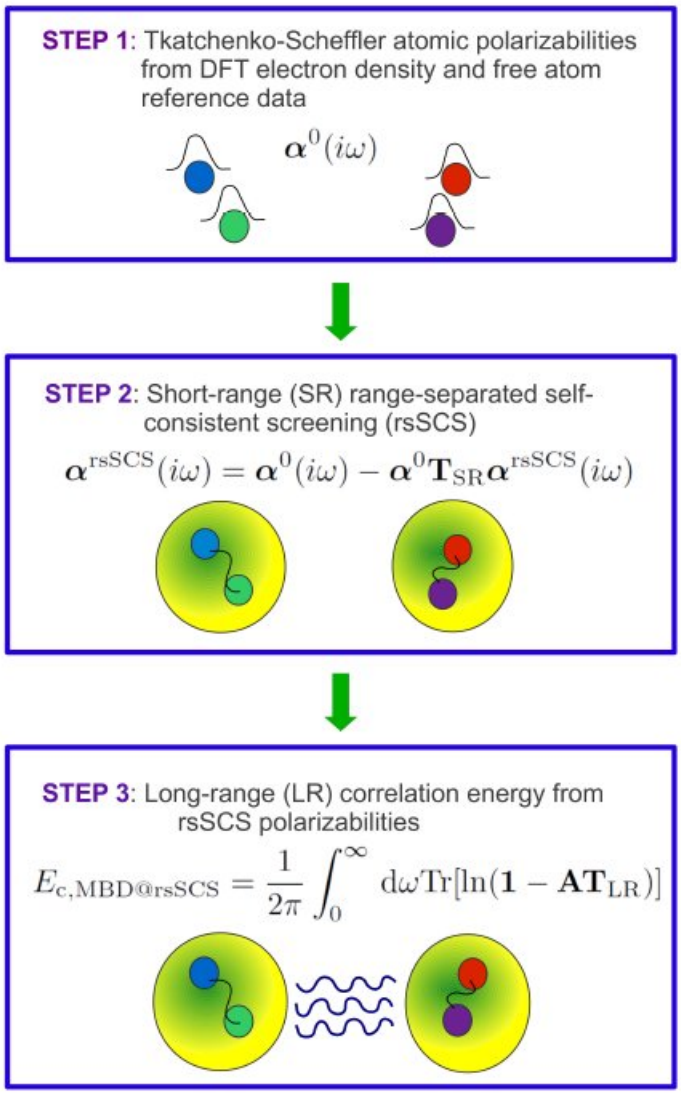


Figure 3.5: Schematic description of the MBD@rsSCS method.

MBD@rsSCS In the range-separated self-consistently screened (rsSCS) version of MBD (MBD@rsSCS) [4], the vdW energy is calculated in three steps (Figure 3.5). First, the free-atom reference vdW parameters (polarizabilities, C_6 coefficients, vdW radii) are scaled with the ratio of Hirshfeld volumes of atoms in the system and of the respective free atoms. Second, the self-consistent screening equation for the dipole oscillators is solved to account for short-range screening between the atoms. Third, the

MBD Hamiltonian is solved to obtain the vdW energy.

MBD-NL The more recent nonlocal (NL) version of MBD (MBD-NL) incorporates several ingredients from the class of nonlocal vdW density functionals to extend the applicability of MBD@rsSCS to ionic and hybrid metal-organic systems [104]. Compared to MBD@rsSCS, the atomic vdW parameters are obtained by coarse-graining a modified Vydrov–Van Voorhis polarizability functional, and the self-consistent screening step is skipped.

Analytical forces and stress are implemented for both versions of MBD, with the following caveat. The total nuclear derivatives of the MBD energy consist of the direct terms and of the implicit terms arising from the dependence on the vdW parameters, which in turn depend either on Hirshfeld volumes (MBD@rsSCS) or the polarizability functional (MBD-NL), which depend on the electron density, which finally depends on nuclear coordinates. Currently, the latter implicit terms are neglected, however, preliminary testing suggests that these terms are negligible in most systems. In fact, the error arising from neglecting these terms seems to be in general comparable to or smaller than the inherent numerical noise in the Kohn–Sham forces. Having said that, please report any observed discrepancies in the forces during geometry relaxations or MD simulations to dev@jan.hermann.name.

Tags for general section of `control.in`:

The following two keywords activate either the MBD@rsSCS (`many_body_dispersion`) or the MBD-NL (`many_body_dispersion_nl`) method. The optional arguments are shared by both methods.

Tag: `many_body_dispersion` (control.in)

Usage: `many_body_dispersion [option=value...]`

Tag: `many_body_dispersion_nl` (control.in)

Usage: `many_body_dispersion_nl [option=value...]`

- `beta=<real>` [default: depends on XC functional] sets the damping parameter β.
- `k_grid=<integer>:<integer>:<integer>` [default: taken from `k_grid`] specifies the k-point grid used for sampling the first Brillouin zone in the MBD calculation. The grid is shifted by half a distance between the k-points from the Γ-point. [only for periodic systems]
- `freq_grid=<integer>` [default: 15] controls the size of the imaginary-frequency grid used for the Casimir–Polder integral.

- `self_consistent=<logical>` [default: `.false.`] turns on self-consistency as explained under `vdw_correction_hirshfeld_sc`. Currently, this works only for MBD@rsSCS.

- `vdw_params_kind=<string>` [default: `"ts"`] specifies the set of free-atom vdW parameters (α_0, C_6, R_{vdW}) used. `"ts"` uses the original set of parameters, `"tssurf"` uses values from the so-called vdW$^{\text{surf}}$ approach [206] for some of the elements. The used parameters are listed here. This option should be left unchanged with MBD-NL, which is designed to work with the default set of vdW parameters in all cases.

Examples:

- `many_body_dispersion` (this uses the default settings)

- `many_body_dispersion_nl beta=0.8 k_grid=3:3:3` (explicit settings of the damping parameter and of a $3 \times 3 \times 3$ k-point grid)

Caveats:

- The `many_body_dispersion` approach, i.e., the MBD@rsSCS approach, should work reliably for dispersion bonded systems. However, the associated Hamiltonian can have negative eigenvalues for systems that are more strongly bonded, typically ionic or metallic systems. If this happens, this would be the result of too high polarizability as predicted by the Hirshfeld volumes. The code will stop if this happens.

- The `many_body_dispersion_nl` is a more advanced scheme and is expected to do a much better job of not overpolarizing parts of the system. Negative eigenvalues should occur much less frequently (hopefully only in corner cases).

- If negative eigenvalues of the MBD Hamiltonians occur, the best path forward is an assessment if this particular approach to include dispersion contributions is physically appropriate for the system in question.

Deprecated FHI-aims implementation (pre-2019 default)

Tag: `many_body_dispersion_pre2019` (control.in)

Usage: `many_body_dispersion_pre2019 [option=value...]`

Purpose: Calculates the MBD@rsSCS energy for the active XC functional (available for PBE, PBE0, and HSE).

This was the default method prior the official 2019 FHI-aims release. The default has changed to the Libmbd implementation `many_body_dispersion` described above.

- `k_grid=<nk1>:<nk2>:<nk3>` [default: taken from `k_grid`] specifies the k-point grid used for sampling the first Brillouin zone in the MBD calculation. The grid is shifted by half a distance between the k-points from the Γ-point. [only for periodic systems]
- `vacuum=<a>:<b>:<c>` [default: all `.false.`] controls whether some of the lattice vectors correspond to vacuum dimensions.
- `self_consistent=<logical>` [default: `.false.`] controls the calculation of the MBD XC potential.
- `beta=<real>` [default: depends on XC functional] sets the damping parameter β.

3.21 Exchange-hole dipole moment (XDM) dispersion method

The exchange-hole dipole moment (XDM) model[21] is a method to calculate the dispersion energy in molecules and solids. The dispersion energy in XDM is calcualted as a damped asymptotic expression:

$$E_{\text{disp}} = -\sum_{i>j} \frac{C_{6,ij} f_6(R_{ij})}{R_{ij}^6} + \frac{C_{8,ij} f_8(R_{ij})}{R_{ij}^8} + \frac{C_{10,ij} f_{10}(R_{ij})}{R_{ij}^{10}} \tag{3.54}$$

where i and j run over atoms, and R_{ij} is the interatomic distance. The XDM contribution is used to correct the energy from the base density functional approximation for missing dispersion:

$$E_{\text{total}} = E_{\text{base}} + E_{\text{DFT}} \tag{3.55}$$

The dispersion coefficients are calculated from the self-consistent density and kinetic energy density. For instance, the leading dispersion coefficients are:

$$C_{6,ij} = \frac{\alpha_i \alpha_j \langle M_1^2 \rangle_i \langle M_1^2 \rangle_j}{\langle M_1^2 \rangle_i \alpha_j + \langle M_1^2 \rangle_j \alpha_i} \tag{3.56}$$

where $\langle M_1^2 \rangle_i$ are the multipole moments of the electron plus exchange-hole dipole distribution and α_i are the in-molecule atomic polarizabilities. The damping functions f_n ($n = 6, 8, 10$) contain two adjustable parameters a_1 and a_2, which are determined for each functional with which XDM is coupled by fitting to a small set of molecular gas-phase dimers. More details can be found in the original reference[21], the periodic XDM implementation[180], and recent reviews[124].

Tags for general section of `control.in`:

Tag: `xdm` (control.in)

Usage: `xdm a1 a2 variable_c6`

Purpose: activates the calcualtion of the XDM dispersion energy and its derivatives.
Options:

- `a1 a2`: XDM dispresion coefficients (a_2 in units of Å).
  Default: the default damping coefficients (a1 and a2) are chosen based on the functional used (see the `xc` keyword), and correspond to a "light" basis set with a dense grid. The dispersion coefficients for the following functionals are available: B86bPBE, B86bPBE-25, B86bPBE-50, PBE, PBE0, PBE50, HSE06. If your functional is not on this list, you need to supply the XDM parameters.

- `variable_c6`: in a geometry relaxation, recompute the dispersion coefficients every step.
  Default: the dispersion coefficients are calculated in the first and last step of a relaxation.

Examples:

- `xdm` (calculate XDM with default settings)
- `xdm 0.4572 0.5921` (use $a_1 = 0.4572$ and $a_2 = 0.5921$Å)
- `xdm variable_c6` (recalculate XDM coefficients every relaxation step)

3.22 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. 3.22.1: One version that allows post-processing only (i.e., compute the self-consistent density by another XC functional, the 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. 3.22.2: 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.

3.22.1 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. [59]. This task follows exactly the recipe presented in the original paper [59]. 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. [172] 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 [59], 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` the 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 [59]. 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`

3.22.2 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 [59] 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 [89]. 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 targer 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.

3.23 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 Hartree-Fock and hybrid functionals, particularly for their periodic implementations, see also the dedicated next section, Sec. 3.24.

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. We hope to provide resilient, system-independent default settings, but there is a lot of freedom beyond those defaults to either tighten up things or speed up calculations (at the price of reduced accuracy).

The present section describes all numerical settings for the aforementioned exchange-correlation treatments.

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, analytic continuation from the imaginary to the real axis and contour deformation 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. This threshold is a value to be tested in case something unexpected happens.

There are four key references that provide the technical background for these sections:

1. Principle of how we calculate the two-electron Coulomb operator by so-called resolution of identity: Xinguo Ren *et al.* (2012), New J. Phys. 14, 053020, Ref. [196]. The approach summarized below and implemented in Keyword `RI_method V` is still the default for any non-periodic many-body perturbation calculations *beyond* DFT (i.e., for MP2, RPA, GW, etc.)
2. Localized resolution of identity (Keyword `RI_method LVL`), which is used by default for all Hartree-Fock and hybrid functional calculations, and which is the only option for any periodic calculations including the two-electron Coulomb operator: Arvid Ihrig *et al.* (2015), New J. Phys. 17, 093020, Ref. [116].
3. Linear-scaling and periodic implementation of periodic Hartree-Fock and hybrid functionals based on `RI_method LVL`, described in Sergey Levchenko *et al.* (2015), Comput. Phys. Commun. 192, 60-69, Ref. [154].

4. Technical details of self-energy evaluation in GW related methods, described in Ref. [196] and more detailed in Golze *et al.* (2018), J. Chem. Theory Comput., *14*, 4856 [82]. The latter contains also a comparison of different techniques.

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_{basis}^4 many 4-center integrals are reduced to $N_{\text{basis}}^2 \cdot N_{\text{aux}}$ many 3-center integrals and N_{aux}^2 many 2-center integrals (where N_{basis} and N_{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" [230] versions which have been implemented in this code. In the "V" version, the 4-center integrals are approximated by

$$(ij|i'j') \approx \sum_{\mu\nu}(ij|\mu)V_{\mu\nu}^{-1}(\nu|i'j'), \tag{3.57}$$

and in the "SVS" one,

$$(ij|i'j') \approx \sum_{\mu\nu}\sum_{\mu'\nu'}(ij\mu)S_{\mu\mu'}^{-1}V_{\mu'\nu'}S_{\nu'\nu}^{-1}(\nu i'j'), \tag{3.58}$$

where $i, j, i', j', \ldots$ denote the regular basis functions and $\mu, \nu, \ldots$ 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_μ respectively. Eq. (3.57) and (3.58) are often refered to as resolution of identity (Refs. [32, 2, 230, 65] and others). In practice satisfactory accuracy can be gained with an auxiliary basis size N_{aux} of 4-5 times of N_{basis}. In addition, we implement a modified localized version of RI-V known as "RI-LVL", described in Ref. [116] and also in Sec. 3.24.

How is the auxiliary basis functions constructed? In FHI-aims, it is built up as the "on-site" pair products of the regular basis functions (hence the auxiliary basis is also called 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

$$\{\xi_{\text{at},n,l}(r)\} = \{u_{\text{at},n_1,l_1}(r)u_{\text{at},n_2,l_2}, |l_1 - l_2| \le l \le |l_1 + l_2|\}. \tag{3.59}$$

In Eq. (3.59) 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 true. 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 orthormalization. The process is described in Ref. [196] and in more detail with an illustrating figure in Ref.

[116] (open access). The parameters that controls the construction of the auxiliary basis functions can be found below.

GW: Self-energy evaluation

Practical guidelines how to conduct GW calculations and a summary of numerical techniques are given a recent, comprehensive review article [80]. In GW, we have to calculate the self-energy Σ, which is given by

$$\Sigma(\mathbf{r},\mathbf{r}',\omega) = \frac{i}{2\pi}\int d\omega' e^{i\omega'\eta} G_0(\mathbf{r},\mathbf{r}',\omega+\omega')W_0(\mathbf{r},\mathbf{r}',\omega'). \tag{3.60}$$

The frequency integration in Eq. (3.60) presents one of the major challenges in a G_0W_0 calculation because G_0 and W_0 have poles close to the real frequncy axis. Different numerical techniques were developed, which are summarized in Ref. [80]. In FHI-aims, the self-energy can be calculated with the analytic continuation and the contour deformation. The contour deformation is computationally more expensive, but more accurate. The implementation of the analytic continuation in FHI-aims is described in [196] and the implementation of the contour deformation in [82].

Analytic continuation

When using the analytic continuation, the self-energy is first calculated on the imaginary frequency axis

$$\Sigma(\mathbf{r},\mathbf{r}';i\omega) = \frac{i}{2\pi}\int d\omega' G(\mathbf{r},\mathbf{r}';i\omega+i\omega')W(\mathbf{r},\mathbf{r}';i\omega') \tag{3.61}$$

and then analytically continued to the real frequency axis. A proper frequency grid is needed for the analytic continuation, i.e., a set of imaginary frequencies $\{i\omega\}$ for which $\Sigma(i\omega)$ is computed. One popular way of performing the analytical continuation is to model the self-energy with a multi-pole expression [203], namely,

$$\Sigma(i\omega) \approx A_0 + \sum_n \frac{A_n}{i\omega - B_n}, \tag{3.62}$$

where n is the number of poles, and A_n and B_n are complex numbers. Eq. (3.62) is used to fitted the calculated self-energy on imaginary axis using the non-linear least square fitting algorithm. Once the 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

$$\Sigma(\omega) \approx A_0 + \sum_n \frac{A_n}{\omega - B_n}. \tag{3.63}$$

In practice $n = 2$ (the so-called two-pole fitting) is often found to give good performance.

A Pade approximation based variant with more poles is also implemented in FHI-aims. In the Pade approximation, the self-energy is parameterized as

$$\Sigma(i\omega) = \cfrac{a_1}{1+\cfrac{a_2(i\omega - i\omega_1)}{1+\cfrac{a_3(i\omega - i\omega_2)}{\cdots}}}. \tag{3.64}$$

For a given, chosen set of calculated self-energy data points $\{i\omega_n, \Sigma(i\omega_n)\}$ with $n = 1, \cdots, N$, the N complex parameters $a_1, \cdots, a_N$ can be uniquely determined. The self-energy on the real frequency axis is then obtained by replacing $i\omega$ by ω in Eq. (3.64). We note that the Pade approximation given by (3.64) can be interpreted as a multipole expression, with the number of poles $N_{pole} = N - 1$.

The type of the analytical continuation used in GW (Eqs. (3.63) or (3.64)) is determined using the `anacon_type` and `n_anacon_par` keywords. The number of frequency points used on the imaginary axis (this determines the accuracy of the input used for fitting the expressions Eqs. (3.63) or (3.64)) can be set using the `frequency_points` keyword. The frequency grid used is a modified Gauss-Legendre grid that ranges from zero to infinity. Our experience suggests that highly accurate results for molecules (few meV accuracy for electronic excitations, compared to exact expressions for the self-energy on the real axis) can be obtained using a 16-parameter Pade approximation with 200 frequency points [232]. However, the numerical accuracy is then much higher (and much more costly) than the accuracy of the underlying GW approximation itself, and thus somewhat reduced defaults are set in the code (see keyword descriptions below).

More importantly, the Pade approximation is also numerically less stable than the two-pole approximation. This means that, for some systems with a complicated pole structure of the self-energy, the Pade fit might not converge for certain eigenvalues. The results must therefore be inspected carefully, even if (for normal light-element molecules and valence-like states) its accuracy can be much higher than the two-pole approximation. This is, in fact, not a simple implementation issue but rather one that goes back to the mathematical structure of the true self-energy.

Contour deformation

The analytical continuation becomes increasingly inaccurate for deeper states since the structure of the self-energy is typically more complicated, i.e., has more poles. The fitted pole models fail to represent these more complicated structures, see Ref. [82]. In these cases, the self-energy should be calculated on the real-frequency axis using the contour deformation technique, where the correlation part of the self-energy is then expressed as

$$\begin{aligned}\Sigma^c(\boldsymbol{r}, \boldsymbol{r}', \omega) = & -\frac{1}{2\pi}\int_{-\infty}^{\infty} d\omega' G_0(\boldsymbol{r}, \boldsymbol{r}', \omega + i\omega') W_0^c(\mathbf{r}, \mathbf{r}', i\omega') \\ & - \sum_i \phi_i(\boldsymbol{r})\phi_i(\boldsymbol{r}') W_0^c(\boldsymbol{r}, \boldsymbol{r}', |\epsilon_i - \omega| + i\eta)\theta(\epsilon_i - \omega) \\ & + \sum_a \phi_a(\boldsymbol{r})\phi_a(\boldsymbol{r}') W_0^c(\boldsymbol{r}, \boldsymbol{r}', |\epsilon_a - \omega| + i\eta)\theta(\omega - \epsilon_a)\end{aligned} \tag{3.65}$$

θ is the Heaviside step function, $\{\phi_n\}$ molecular orbitals and $W_0^c(\mathbf{r}, \mathbf{r}', \omega) = W_0(\mathbf{r}, \mathbf{r}', \omega) - v(\mathbf{r}, \mathbf{r}')$ with the bare Coloumb interaction $v(\mathbf{r}, \mathbf{r}')$. The index i refers to occupied and a to unoccupied orbitals. The evaluation of the self-energy with the contour deformation technique is controlled by the keyword `contour_def_gw`. An example input file for GW with the contour deformation can be found in [79].

Tags for general section of `control.in`:

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), of 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 matrx 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: `anacon_type` (control.in)

Usage: `anacon_type string`

Purpose: 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)
`string` is a string that indicates the self-energy type, either 'two-pole' or 'pade'.
Default: No default (must be set by the user if the self-energy is required).

- `string` = 'two-pole' or '0' : The normal two-pole fitting (Eq (3.63)).
- `string` = 'pade' or '1' : Pade approximation (Eq. (3.64)).

The number of parameters in either approximation can be set using the keyword `n_anacon_par` .

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.

This keyword must be set in control.in if the self-energy on the real axis is needed (usually, for GW). In past versions of FHI-aims, the code would set a silent default for `anacon_type` if a self-energy calculation for the real axis was required. This is no longer the case in present versions. Users must make this choice explicitly in 'control.in';

if that is not the case, the code will stop with a (hopefully gentle and instructive) warning message.

The reason is that the choice of the analytical continuation used can have a noticeable effect on the accuracy of GW-calculated eigenvalues. The two-pole approximation is well established, but less accurate than the Pade approximation when the latter works.[232] On the other hand, systems with a complicated self-energy structure can lead to numerical problems with the Pade approximation that can result in seemingly random values for certain predicted quasiparticle eigenvalues (this can be tested, for instance, by modifying the `frequency_points` keyword and tracking the results).

Tag: `freq_grid_type` (control.in)

Usage: `freq_grid_type value`

Purpose: If set, specifies the type of the grid for the imaginary frequency.

- `value` = 0 : Normal Gauss-Legendre grid ranging from 0 to a maximum frequency, specified by the keyword `maximum_frequency` .
- `value` = 1 : Modified Gauss-Legendre grid ranging from 0 to positive infinity.
- `value` = 2 : Logrithmic grid ranging from 0.01 a.u. to a maximum value specified by `maximum_frequency` .

Default: `value`=1

Note that the `freq_grid_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: `n_anacon_par` (control.in)

Usage: `n_anacon_par value`

Purpose: If set, specifies the number of parameters used in the two-pole fitting (Eq. (3.62)) or Pade approximation (Eq (3.64)). The default value for `n_anacon_par` is 5 if `anacon_type` is set to 'two-pole' (two-pole fitting), and 16 if `anacon_type` is set to 'pade' (Pade approximation).

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: `contour_def_gw` (control.in)

Usage: contour_def_gw $\texttt{state}_{\texttt{start},\alpha}$ $\texttt{state}_{\texttt{end},\alpha}$ $\texttt{state}_{\texttt{start},\beta}$ $\texttt{state}_{\texttt{end},\beta}$

Purpose: If set, specifies the range of states for which the GW quasiparticle energies are computed with the contour deformation, see Ref. [82] for a description of the implementation. The range can be specified for the α and β spin channel separately. Giving the range for the β channel is optional. If not specified, the range set for α will be also used for the β channel. For spin-unpolarized calculations specify only the α channel.
The quasiparticle energies for the other states are computed with the analytic continuation, i.e., the parameters anacon_type and n_anacon_par should be set as well. Setting frequency_points is mandatory (200 grid points is a solid choice).

Tag: `contour_spin_channel` (control.in)

Usage: contour_spin_channel integer (1 or 2)

Purpose: If specified, restricts the contour deformation to a certain spin channel. If not given, QP energies for both channels will be calculated.

Tag: `contour_eta` (control.in)

Usage: contour_eta real

Purpose: Specifies the broadening parameter η used for the contour deformation. It might be useful to set this parameter to higher values (e.g 0.002 a.u.) when printing the self-energy or spectral function. Otherwise the default setting ensures numerical accuracy and stability.
Default: 0.001 a.u.

Tag: `contour_restart` (control.in)

Usage: contour_restart task

Purpose: The iteration of the QP equation can become expensive for large systems. A restart is possible.

task is a string, specifying the desired restart task.

Available options for task are:

- write : Writes restart info to file contour_gw_qp_energies.dat.
- read : Reads restart info from contour_gw_qp_energies.dat and continues the QP iteration cycle.
- read_and_write : Performs what the write and read options do. If the restart files do not exist, the code will still proceed normally.

Tag: `full_cmplx_sigma` (control.in)

Usage: `full_cmplx_sigma boolean`

Purpose: Technical keyword for the contour deformation. If set to .true., the complex broadening term $i\eta$ enters also the integral term of the self-energy. This requires more grid points in the frequency integration, i.e., `frequency_points` should be set to 2000. Including $i\eta$ is not necessary when calculating the quasiparticle energies, but gets rid of (the very small) unphysical steps in the spectral function at the KS/HF energies.
Default: .false.

Tag: `gw_zshot` (control.in)

Usage: `gw_zshot boolean`

Purpose: If set to .true., the Z-factor is calculated and the quasiparticle equation is not calculated iteratively, but linearized using a Taylor expansion. Less exact than the iterative solution. Works with analytic continuation an contour deformation.
Default: .false.

Tag: `contour_zshot_offset` (control.in)

Usage: `contour_zshot_offset real`

Purpose: When using `gw_zshot` , the Z-factor is calculated, which contains the dervative of the self-energy with respect to the frequency. For the contour deformation, this derivative is calcuated numerically. This keyword defines the offset (delta value) to calculate the derivate numerically and should be a small number.
Default: 0.002 a.u.

Tag: `gw_hedin_shift` (control.in)

Usage: `gw_hedin_shift boolean/state`

Purpose: If set, the poor-man's self-consistency proposed by Hedin [98] is enabled. The Hedin shift is referenced to a particular state, typically the HOMO. This procedure can be also considered as fixing the zero of the energy scale in a G_0W_0 calculation employing an overall energy shift ΔE. When including this shift, the starting point dependence is significantly reduced, similar to an ev-scGW_0 calculation. The computational overhead is negligible. The Hedin shift can be calculated individually for each state. In this case, set '.true.' The shift can be also calculated for a particular state, which is then applied to all other states. For the latter, give an integer for the state instead of a boolean. Note that this is the common way to apply the Hedin shift with the HOMO as reference level.
Default: .false.

Tag: `post_adjust_qp_relativistic` (control.in)

Usage: post_adjust_qp_relativistic $\texttt{level}_1$ $\texttt{adjustment}_1$, ... , $\texttt{level}_n$ $\texttt{adjustment}_n$

Purpose: At the conclusion of the quasiparticle calculation, the nth quasiparticle level is adjusted by the given amount (in eV). Relativistic effects are very pronounced for the deep core states, but these effects are rather independent of the valence configuration, so for the 1s core of light elements a post-quasiparticle correction dependent only on the atomic species of the orbital provides sufficient accuracy. Only works in conjunction with contour deformation. See reference [127] for further details.

Tag: `pre_adjust_qp_relativistic` (control.in)

Usage: post_adjust_qp_relativistic $\texttt{level}_1$ $\texttt{adjustment}_1$, ... , $\texttt{level}_n$ $\texttt{adjustment}_n$

Purpose: As for post_adjust_qp_relativistic , but the adjustment is made to the KS-orbital energies before the quasiparticle calculation is initiated.

Tag: `print_self_energy` (control.in)

Usage: print_self_energy state $\texttt{freq}_{\texttt{start}}$ $\texttt{freq}_{\texttt{end}}$

Purpose: If set, the diagonal self-energy matrix elements for the requested state are printed in the given frequency range that should be given in eV. Setting the frequency range is optional.

Tag: `calc_spectral_func` (control.in)

Usage: calc_spectral_func $\texttt{freq}_{\texttt{start}}$ $\texttt{freq}_{\texttt{end}}$ resolution

Purpose: If set, the total spectral function $A(\omega)$ is printed in the given frequency (ω) range that should be given in eV. The spectral function is defined as

$$A(\omega) = \frac{1}{\pi} \sum_m \frac{|\Im\Sigma_m(\omega)|}{\left[\omega - \varepsilon_m - (\Re\Sigma_m(\omega) - v_m^{xc})\right]^2 + \left[\Im\Sigma_m(\omega)\right]^2} \tag{3.66}$$

where m runs over all occupied and virtual states and where we include also the imaginary part of the complex self-energy Σ. See reference [81] for further details. Implemented for G_0W_0, evGW_0 and evGW in combination with contour deformation. Unlike for fully self-consistent GW, we include only the diagonal matrix elements of Σ. Note that the calculation of the spectral is computationally much more expensive than solving the QP equation and is not recommended for prodcution runs. However, it gives access to the underlying physics, e.g., to investigate multisolution behavior and peak intensities.

Setting the resolution is optional. If not given, the resolution is 0.001 eV.

Tag: `spectral_func_state` (control.in)

Usage: `spectral_func_state state`

Purpose: If set, the spectral function is only calculated for state n

$$A_n(\omega) = \frac{1}{\pi} \frac{|\Im\Sigma_n(\omega)|}{\left[\omega - \varepsilon_n - (\Re\Sigma_n(\omega) - v_n^{xc})\right]^2 + \left[\Im\Sigma_n(\omega)\right]^2} \tag{3.67}$$

where the total spectral function defined in Eq. (3.66) is $A(\omega) = \sum_m A_m(\omega)$.

Keyword is only active if `calc_spectral_func` is set.

Tag: `iterations_sc_cd` (control.in)

Usage: `iterations_sc_cd integer`

Purpose: Sets the maximum number of iterations in the eigenvalue self-consistent loop in an evGW (ev_scgw) or evGW_0 (ev_scgw0) calcuation using the contour deformation (`contour_def_gw`). Note, this keyword does not set the number of iterations to converge the QP equation, but the number of iterations in the outer loop (iteration of eigenvalues in G and W). Convergence of the outer loop is usually reached within 10-20 steps.

Default: 20

Tag: `nocc_sc_cd` (control.in)

Usage: `nocc_sc_cd integer`

Purpose: Sets the number of occupied states that enter in the eigenvalue self-consistent loop in an evGW (ev_scgw) or evGW_0 (ev_scgw0) calcuation using the contour deformation (`contour_def_gw`). Ideally all states should enter, but such a calculation is expensive with the contour deformation. If set to, e.g, 3, the first three occupied states (HOMO, HOMO-1 and HOMO-2) will enter in addition to the ones specified in `contour_def_gw` . A scissor shift will be applied to the rest of the occupied orbitals. It is recommend to include *all* occupied states if possible.

Default: 5

Tag: `nvirt_sc_cd` (control.in)

Usage: nvirt_sc_cd integer

Purpose: Sets the number of virtual states that enter in the eigenvalue self-consistent loop in an evGW (ev_scgw) or evGW_0 (ev_scgw0) calcuation using the contour deformation (contour_def_gw). Ideally, all states should enter, but such a calculation is expensive with the contour deformation. If set to, e.g, 3, the first three unoccupied states (LUMO, LUMO+1 and LUMO+2) will enter in addition to the ones specified in contour_def_gw . A scissor shift will be applied to the rest of the virtual orbitals. It is recommended to include only 5 to 10 virtual states explicitly, in particular if the states of interest are, e.g., core states.

Default: 5

Tag: `sc_reiterate` (control.in)

Usage: sc_reiterate boolean

Purpose: Keyword for evGW (ev_scgw) or evGW_0 (ev_scgw0) calcuations using the contour deformation (contour_def_gw). States that have converged in the eigenvalue self-consistent (outer) loop exit the eigenvalue iteration in G and W. If this keyword is set and if the states given in contour_def_gw converge before convergence of the evGW or evGW_0 calculation is reached, they are re-iterated at the end. The changes upon re-iteration is typically smaller than 0.1 eV. Re-iteration is recommended for calculations that require benchmark accuracy.

Default: .false.

Tag: `try_zshot` (control.in)

Usage: try_zshot boolean

Purpose: Keyword for evGW (ev_scgw) or evGW_0 (ev_scgw0) calcuations using the contour deformation (contour_def_gw). The QP solution might not converge for core states or high-energy virtual states when using a GGA starting point for the G_0W_0 calculation due to a lack of a distinct QP peak, which has been explained in [81]. For the first few iterations of the evGW or evGW_0 calculation, the QP equation for some states might thus not converge. If this keyword is set to true, an approximation of the non-converged QP energies is obtained by linearizing the QP equation (Z-shot). Check that the Z-shot solution is not used in the last iterations of the evGW and evGW_0 calculation since it is numerically not very exact, in particular for deep states.

Default: .true.

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 lvl`, 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 lvl`, 5 for `RI_method V` and nuclear charge $Z <= 54$, and 6 for `RI_method V` and $Z > 54$.

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 lvl`, 6 otherwise.

Tag: `frequency_points` (control.in)

Usage: `frequency_points value`

Purpose: If set, specifies the number of (imaginary) frequency points for the self-energy calculation.

The default value for the frequency points depends on the choice of the analytical continutation type. For two-pole fitting (`anacon_type` ='two-pole'), the default value for `frequency_points` value is 40; for the Pade approximation (`anacon_type` ='pade') the default value for `frequency_points` is 100.

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: `maximum_frequency` (control.in)

Usage: `maximum_frequency value`

Purpose: If set, specifies the maximal (imaginary) frequency value for the self-energy self-consistent calculation. The unit for `value` here is Hartree.

Note that the `maximum_frequency` only makes sense when the `freq_grid_type` is set to be 0 or 2, i.e., when the stdandard Gauss-Legendre grid or logrithmic grid is used. For `freq_grid_type` =0, the default value for `maximum_frequency` is 10 Hartree; for `freq_grid_type` =2, the default value is 5000 Hartree. However, when self-consistent GW is involked (both scGW and scGW_0), the default value for `maximum_frequency` is 7000 Hartree.

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^{-1}. The default value is 1000 a.u..

Note that the `maximum_time` only makes sense if `sc_self_energy` is set, i.e., the 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: `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 taylored 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:` $\min(16, \lfloor N_{\mathrm{aux}}/N_{\mathrm{proc}} \rfloor)$.

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 noticable 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: `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, with possible options listed below.

Default: Non-periodic: `type=lvl` for Hartree-Fock and hybrid functional calculations, which can be used for geometry relaxations. `type=v` for MP2, RPA and GW etc. calculations that require unoccupied states. This gives better accuracy for a given auxiliary basis, but is usually more expensive and provides no geometry relaxation at present. Periodic: `type=LVL_fast`, which provides the necessary reduction to what is essentially an $O(N)$ framework.

Different options for the `type` option include:

- `svs` (for the "SVS" version, i.e., Eq. (3.58))
- `v` (for the "V" version, i.e. Eq. (3.57))
- `lvl` (for the "LVL" version)
- `LVL`, `LVL_fast`, or `lvl_fast` are all synonymous with option `lvl`
- `lvl_full` implements a non-linear scaling version of the "LVL" approach for total energy evaluations only
- `lvl_2nd` implements the so-called "robust Dunlap correction" for total energies only, which essentially follows up an s.c.f. calculation with an additional RI-V-like step. In our experience, similar results are better accomplished without this correction (see also Ref. [116]).

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 Hartree-Fock and hybrid functional calculations 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. 3.24 and in Refs. [116, 154]. It also relies extensively on screening near-zero elements of the density matrix and of the Coulomb operator. On the other hand, the localized version has slightly larger errors than RI-V for hybrid functionals, and *significantly* larger errors for correlated methods like MP2 or RPA. These can be remedied by adding a few extra functions to the construction of the auxiliary basis set using the `for_aux` keyword, as described in detail in Ref. [116].
- At present, *do not use RI-LVL for MP2 and RPA* unless you know what you are doing. It is possible to repair their accuracy in RI-LVL, by increasing the size of the auxiliary basis set using the `for_aux` keyword, as described in detail in Ref. [116], but please see the figures and benchmarks in that reference before trying.
- `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, `lvl_2nd`, which adds a correction term to the nonlocal exchange energy to make it "robust" in the Dunlap sense [62], that is, the error in the energy is quadratic in the error in the product expansion.

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 4096.

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: 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: use_logsbt (control.in)

Usage: use_logsbt bool

Purpose: If set, the two-center integrals are calculated by one-dimensional integrations in Fourier-space. In the case of RI_method LVL, also the three-center integrals are computed by this method.

Default: .true.

use_logsbt .true. is faster and more accurate than the alternative.

The algorithm for the overlap and Coulomb integrals is described by Talman in [221]. It uses an efficient spherical Bessel transform on a logarithmic radial grid [222, 91, 92] 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^{-2}].
`alpha_i coeff_i` : *If* `N`>1, $i = 1, \ldots, N$ additional lines specify exponents α_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.

Adds extra basis function ONLY to the construction of the auxiliary basis set used to expand the Coulomb operator (resolution of identity, see keyword `RI_method`). In particular, the accuracy of `RI_method LVL` can be increased by adding extra high-angular momentum radial functions to the auxiliary basis set. The improvement becomes less and less relevant as the orbital basis set itself increases. For instance, there may be a noticeable change for *tier 1*, but much less or not at all for *tier 2*. Currently supports only `hydro` and `ionic` basis functions.

For a detailed description with benchmarks, please see Ref. [116].

Here is an example for the use of the `for_aux` , which was obtained by altering the "light" settings of the C atom:

```
[...]
#  "First tier" - improvements: -1214.57 meV to -155.61 meV
     hydro 2 p 1.7
     hydro 3 d 6
     hydro 2 s 4.9
```

```
#  "Second tier" - improvements: -67.75 meV to -5.23 meV
  for_aux     hydro 4 f 9.8
#     hydro 3 p 5.2
#     hydro 3 s 4.3
  for_aux     hydro 5 g 14.4
#     hydro 3 d 6.2
[...]
```

Adding those two high-angular momentum functions does improve the quality of the RI_method LVL noticeably, at the price of more CPU time. On the other hand ... "light" settings are specifically chosen because not everything is completely converged. In fact, the orbital basis set error itself may be larger than the accuracy gained by amending the two-electron Coulomb operator expansion.

---

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.

3.24 Hartree-Fock and hybrid functionals, including periodic systems

Periodic versions of the Hartree-Fock method and of and hybrid density functionals are implemented in FHI-aims. Generally, our experiences are very good. The implementation is stable and seems to scale well towards large systems. However, we still ask you to exercise some care. If you encounter any unexpected difficulties, consult the developers.

Periodic versions of MP2, RPA and GW are not yet part of the main FHI-aims distribution, as they are in various stages of development. Please feel free to ask at our slack channel regarding these methods for periodic systems. They are very important to us, and we will be happy to share them as they become ready and more usable.

Complete descriptions of the material described below, as well as extensive benchmarks, are summarized in Ref. [154], as well as Ref. [116] (for the non-periodic implementation of the "LVL" approach). Forces and the stress tensor are also implemented, with a clear description of the stress tensor given in Ref. [133].

There are two specific issues from a usability point of view which should be considered:

- The `RI_method` "LVL" described below is implemented in a linear scaling and is therefore the only reasonable pathway for large and/or periodic systems. It is, however, slightly less accurate (for formal reasons) than the non-linear-scaling version for non-periodic systems, RI-V. Please bear this in mind. For standard solids and hybrid functionals, the effects seem very small. See reference [116] for quantitative tests. In general, RI-LVL has been extremely reliable for us.
- For band structure output, the `exx_band_structure_version` keyword allows to toggle between 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 and a slow(!) fallback version that is calculated in reciprocal space. 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. The slow fallback version should not simply be used by default since it can easily become the computational bottleneck – both regarding time and memory.

In periodic Hartree-Fock (and hybrid functional) implementations, the key quantity that needs to be evaluated is the exact-exchange matrix,

$$K_{ij}(\mathbf{k}) = \sum_{kl,\mathbf{q}} D_{kl}(\mathbf{q}) \iint d\mathbf{r} d\mathbf{r}' \frac{\phi^*_{i\mathbf{k}}(\mathbf{r})\phi_{k\mathbf{q}}(\mathbf{r})\phi^*_{l\mathbf{q}}(\mathbf{r}')\phi_{j\mathbf{k}}(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} \tag{3.68}$$

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,

$$K_{ij}(\mathbf{k}) = \sum_{\mathbf{R}} e^{i\mathbf{k}\mathbf{R}} X_{ij}(\mathbf{R}) \,, \tag{3.69}$$

where

$$X_{ij}(\mathbf{R}) = \sum_{kl} \sum_{\mathbf{R}'} D_{kl}(\mathbf{R}') \sum_{\mathbf{R}''} \iint d\mathbf{r} d\mathbf{r}' \frac{\phi_i(\mathbf{r})\phi_k(\mathbf{r}+\mathbf{R}'')\phi_j(\mathbf{r}'+\mathbf{R})\phi_l(\mathbf{r}'+\mathbf{R}'+\mathbf{R}'')}{|\mathbf{r}-\mathbf{r}'|} \tag{3.70}$$

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. (3.68). An alternative, and more efficient implementation is based on the"real-space" formulation, where one first computes $X_{ij}(\mathbf{R})$ from Eq. (3.70), 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. 3.23. 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

$$\phi_i(\mathbf{r}-A_i)\phi_j(\mathbf{r}-A_j) = \sum_{\mu} C^{\mu(A_i)}_{i(A_i),j(A_j)} P_\mu(\mathbf{r}-A_i) + \sum_{\nu} C^{\nu(A_j)}_{i(A_i),j(A_j)} P_\nu(\mathbf{r}-A_j)\,, \tag{3.71}$$

where μ and ν 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: `calculate_fock_matrix_version` (control.in)

Usage: `calculate_fock_matrix_version value`

Purpose: Sets the internal code version to be used for the the linear-scaling evaluation of the exchange matrix.

`value` is a number between 0 (zero) and 5 (five). Default: 5.

A significant optimization effort has targeted the large-scale hybrid density-functional theory part of FHI-aims since the original implementation in Ref. [154]. The original version, `calculate_fock_matrix_version 0`, is still available and went up to seriously large system sizes (up to 1,024 atoms were demonstrated in the original paper) but the updated versions (now the default) are significantly more memory-efficient, time-efficient, and scale to yet much larger systems.

The current default (at the time of writing) is `calculate_fock_matrix_version 5` and offers a fairly sophisticated, load-balanced infrastructure that scales nearly ideally with number of processor for large systems and reduces memory use to the level deemed appropriate for the available compute nodes. It is based on shared memory usage within the MPI-3 standard and notably avoids OpenMP (no OpenMP used and no OpenMP commands should ever be needed with FHI-aims). This allows us to control the near-optimal layout of arrays etc. entirely within the code, for a given number of MPI tasks and compute nodes.

A possible, still rather efficient fallback in case of issues is `calculate_fock_matrix_version 4`, which is less sophisticated (but also less efficient) in its efforts to control load balancing. However, note that the correct way to address a failure of the hybrid DFT in FHI-aims (e.g., due to lack of memory for very large systems) using `calculate_fock_matrix_version 5` is NOT just to fall back to `calculate_fock_matrix_version 4`. Rather (in our experience) one may simply need some more nodes, towards very large systems; alternatively, some underlying MPI libraries (not part of FHI-aims itself) may have unresolved bugs that can be addressed by switching to a different MPI library. Our recommendation is to just stick with `calculate_fock_matrix_version 5` and address the underlying reasons for an observed problem, if helpful by contacting us via the Slack channel.

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. `exx_band_structure_version 1` is preferred (but see below).

The distinction between real-space and reciprocal-space pertains to the method used to calculate the Fock matrix; in both cases, the coordinate system used when specifying the k-path via the `output band` keyword is expressed in terms of reciprocal coordinates.

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 11. We also note that 21 is a reasonable value to sample the fine features of a band structure.
- `exx_band_structure_version 1` is preferred. The real-space band structure version `value`=1 has low overhead and is accurate IF a reasonably dense `k_grid` is used during the preceding s.c.f. calculation. `exx_band_structure_version 1` is therefore the recommended approach. For very sparse s.c.f. `k_grid` settings, it can, however, fail. In that case, the failure is so obvious that one cannot miss it. For better results, please avoid particularly `k_grid` dimensions of 1 (one) in the s.c.f. part of the calculation. We apologize for this inconvenience. On the bright side, if you use `exx_band_structure_version 1` correctly, it will give reliable results without much overhead compared to the underlying s.c.f. calculation. We recommend to test to be sure.
- `exx_band_structure_version 2` is a fallback method that will always work but comes with significant time and memory overhead. If the plotted band structure from the real-space version `exx_band_structure_version` `value`=1 has obvious numerical problems, please switch to a denser `k_grid` during s.c.f. Only if this approach is not successful or possible, consider `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.

We apologize that this decision process is a bit rough around the edges and leaves an essential decision up to the user (because we want you to know). However, consider this:

The above procedure provides a simple way to make band structure output work, and works safely. It is now generally not a problem to produce band structures with hybrid functionals in FHI-aims and this functionality has produced much successful science.

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.` This is, however, very expensive.

Tag: `split_atoms` (control.in)

Usage: `split_atoms flag`

Purpose: The "split_atoms" periodic HF implementation can be switched off by setting `flag` to be `.false.`

This keyword is no longer needed since `calculate_fock_matrix_version` 5 (the current default of the linear-scaling exact exchange implementation for Hartree-Fock and hybrid DFT) effectively supersedes it.

We no longer document this keyword here for this reason. Note that, if you use it with `calculate_fock_matrix_version` 5, `split_atoms` may actually increase the memory usage substantially (potentially preventing a calculation from working) while not offering any other gains.

3.25 Periodic GW in FHI-aims

A periodic version of GW (more precisely the one-shot G_0W_0) has been implemented in FHI-aims. The current implementation is based on the k-space formalism, similar to the k-space implementation of periodic HF (can be invoked by setting `use_hf_kspace` to be true), and only works for insulating systems. So far, tests show that it is rather stable, but if you encounter any problem, consult the developers.

Complete descriptions of the key algorithm and implementation details can be found in Ref. [199]. Here we only present the central equation for computing the periodic G_0W_0 self-energy,

$$\begin{aligned}\Sigma_{n,\sigma}^{G_0W_0}(\mathbf{k}, i\omega) &= \iint d\mathbf{r}d\mathbf{r}'\psi_{n,\sigma}^{\mathbf{k}*}(\mathbf{r})\Sigma_{\sigma}^{G_0W_0}(\mathbf{r},\mathbf{r}', i\omega)\psi_{n,\sigma}^{\mathbf{k}}(\mathbf{r}') \\ &= -\frac{1}{2\pi}\sum_{m,\mathbf{q}}\sum_{\mu,\nu}\int_{-\infty}^{\infty} d\omega' \frac{C_{n,m,\sigma}^{\mu}(\mathbf{k},\mathbf{k}-\mathbf{q})W_{0,\mu\nu}(\mathbf{q}, i\omega')C_{m,n,\sigma}^{\nu}(\mathbf{k}-\mathbf{q},\mathbf{k})}{i\omega - i\omega' + \mu - \epsilon_{m,\sigma}^{\mathbf{k}-\mathbf{q}}}.\end{aligned} \tag{3.72}$$

where the expansion coefficients $C_{n,m,\sigma}^{\mu}(\mathbf{k},\mathbf{k}-\mathbf{q})$ are determined using the "RI-LVL" (or localized RI) approximation, similar to the periodic HF and hybrid function case discssued in Sec. 3.24.

At present, only the one-shot G_0W_0 scheme is implemented for periodic systems. The periodic G_0W_0 can be run in much the same way as the cluster (finite system) case. That is, one needs to specify a starting point by setting `xc` to a desired funcitonal. At the moment, LDA, GGAs and global hybrid functions can be employed as the starting points.

The periodic G_0W_0 is invoked by setting `qpe_calc` to `gw_expt` and setting `k_grid` to appropriate values. Similar to the cluster case, the periodic G_0W_0 self-energy is first computed on imaginary frequencies and then analytically continuted to the real frequency axis. As such, one needs to explicitly specify the type of analytical continuation procedure by setting the `anacon_type` to 0 (two-pole fitting) or 1 (Padé approximation.) The number of frquency points and the number of analytical continuation parameters can be specified by setting `frequency_points` and `n_anacon_par` to appropriate values. If not, the same default values as the clusters will be used for these two parameters. Further details can be found in Sec. 3.23.

As mentioned above, peroidic GW calculations rely on the RI-LVL scheme. This implies one typically needs a larger set of auxiliary basis functions to achieve a satisfactory level of numerical accuracy. In practice, one usually needs to add so-called "for_aux" basis functions in `species_default`, that are used to generate additional auxliary basis functions. Tests show that the following "for_aux" basis setting usually works rather well, and is recommended as a first choice.

```
for_aux     hydro 4 f  3.0
for_aux     hydro 5 g  3.0
```

However, this does not mean the above setting is optimal, or always sufficient. If in doubt, one is encouraged to vary the effective charge parameter Z (the value "3.0" above) to check how the obtained results (e.g., the band gap) change.

The outputs from periodic G_0W_0 calcualtions are the G_0W_0 quasiparticle energies within a window of energy levels at pre-set $\mathbf{k}$ points. The specification of the $\mathbf{k}$ points can be done in two ways. In the first way, one can directly compute the quasiparticle energy band structures along a set $\mathbf{k}$-point paths. The way to set up the $\mathbf{k}$-point paths is exactly the same as in KS-DFT calculations, through the keyword `output band` . In the second way, one can choose to print out the quasiparticle energy level on a sef of $\mathbf{k}$ points belonging to the uniform $\mathbf{k}$ grid, that is used in the preceding SCF calculations. This is set through the keyword `output gw_regular_kgrid` . At each $\mathbf{k}$ point, the set of energy levels for which one asks for quasiparticle energy calculations can be specified through the keyword `state_lower_limit` and `state_upper_limit` , in exactly the same way as in the cluster case. If `state_lower_limit` and `state_upper_limit` are not explicitly specified, G_0W_0 quasiparticle calculations will be done for all the occupied states and a few low-lying unoccupied states.

Tags for general section of `control.in`:

Tag: `qpe_calc` (control.in)

Usage: `qpe_calc gw_expt`

Purpose: If set, and if a finite $\mathbf{k}$ grid is also set via the keyword `k_grid`, peroidc G_0W_0 calcualation will be started.
Here we choose to use a different keyword `gw_expt` instead of `gw` as in the cluster case, in order to emphasize that this is still an experimental version. In future we may change the setting `gw_expt` back to `gw`.

Tag: `output gw_regular_kgrid` (control.in)

Usage: `output gw_regular_kgrid`

Purpose: If set, the quasiparticle energy levels at the regular $\mathbf{k}$ grid will be printed out. However, one does not automatically print out the information on all $\mathbf{k}$ points (which would be often too much), but rather for a number specified through `output k_eigenvalue number`. That is, the $\mathbf{k}$ points where quasiparticle energy levels are printed out are precicely the same as those on which the preceding KS/HF eigenvalues are printed.

Tag: `periodic_gw_optimize` (control.in)

Usage: `periodic_gw_optimize inverse avoid_hang freqbatch:16 blacs_block_size:128 lvlkq_block_size:16 pol_s_block_size:512`

Purpose: Keywords to change the technical details of the periodic GW calculations. `inverse` is to switch on the Cholesky matrix inverse algorithm instead of using the spectrum decomposition algorithm. `avoid_hand` is to use a more stable version of message passing. `freqbatch:xxx` is to use an extra layer of parallization over frequency integration where xxx is the number of sub mpi groups. `blacs_block_size:xxx` is the BLACS block size of the parallel linear algebra. `lvlkq_block_size:xxx` and `pol_s_block_size:xxx` are used to control the batch sizes for g_times_w and polarisability calculations. The default is hard coded to make sure of using 800 MB RAM per MPI.

Tag: `periodic_gw_optimize_single_precision` (control.in)

Usage: `periodic_gw_optimize_single_precision g_times_w polarisability self_energy`

Purpose: Keywords to switch on the single precision calculations in certain region of the code.

Tag: `periodic_gw_optimize_init` (control.in)

Usage: `periodic_gw_optimize_init inverse scale fft`

Purpose: Keywords to switch on the optimized version of the init steps for periodic GW. `fft` requires the code to be linked with an optimized FFT library.

Tag: `periodic_gw_optimize_use_gpu` (control.in)

Usage: `periodic_gw_optimize_use_gpu g_times_w polarisability inverse pxgemm`

Purpose: Keywords to switch on the GPU accelerated on certain steps in periodic GW calculations.

3.26 TDDFT - linear response

These routines are not completed yet. For now it is only possible do use f_{xc} kernels from LDA. The development goes on and more funcionality will be added. When publishing results obtained from this routine, please do cite me, Jan Kloppenburg as the author, as well as of course the usual people in the aims references. When problems, questions or suggestions arise, feel free to contact me at `kloppenburg@fhi-berlin.mpg.de`. Only use there routines if you know what you are doing!

Theory

The goal is to calculate excitation energies ω_I and corresponding oscillator strengths f_I from

$$\mathbf{\Omega}\mathbf{F}_I = \omega_I^2 \mathbf{F}_I \; . \tag{3.73}$$

Linear response theory (see [40]) is the basis for this calculation. We construct

$$\mathbf{\Omega}_{ias,jbt} = \delta_{i,j}\delta_{a,b}\delta_{s,t}(\epsilon_a - \epsilon_i)^2 + 2\sqrt{\epsilon_{as} - \epsilon_{is}}\mathbf{K}_{ias,jbt}\sqrt{\epsilon_{bt} - \epsilon_{jt}} \tag{3.74}$$

with the coupling kernel

$$\mathbf{K}_{ias,jbt} = \int\int \varphi_i^*(\mathbf{r})\varphi_a(\mathbf{r}) \left[\frac{1}{|\mathbf{r} - \mathbf{r}'|} + f_{xc}(\mathbf{r}, \mathbf{r}')\right] \varphi_j^*(\mathbf{r}')\varphi_b(\mathbf{r}')d\mathbf{r}d\mathbf{r}' \; . \tag{3.75}$$

In this notation I refer with the indices i, j to occupied and with a, b to virtual orbitals, while s and t denote the spin. The input energies ϵ are obtained from either ground state *Hartree-Fock* or DFT calculations. Going by this rule we contruct the matrix $\mathbf{\Omega}$ which then is solved for eigenvalues and eigenvectors. The excitation energies ω_I then follow from From the eigenvectors $\mathbf{F}_I$ the oscillator strengths f_I can be obtained from

$$f_I = \frac{2}{3}\omega_I \left[\left| \left\langle \Psi_0 \middle| \hat{\mathbf{X}} \middle| \Psi_I \right\rangle \right|^2 + \left| \left\langle \Psi_0 \middle| \hat{\mathbf{Y}} \middle| \Psi_I \right\rangle \right|^2 + \left| \left\langle \Psi_0 \middle| \hat{\mathbf{Z}} \middle| \Psi_I \right\rangle \right|^2 \right] , \tag{3.76}$$

with the $\hat{\mathbf{X}}$ being the spatial operator for the X direction and the others respectively with Ψ_0 being the all electron ground state wave function and Ψ_I being the all electron excited state wave function for the state I with excitation energy ω_I.

For the TDHF calculation mode, the kernel $\mathbf{K}_{ias,jbt}$ is modified to become

$$\mathbf{K}_{ias,jbt} = (ias|jbt) + \delta_{s,t}(ij|ab) \tag{3.77}$$

that has only the bare *Coulomb* part $(ia|jb)$ and the exact exchange part $(ij|ab)$ from *Hartree-Fock* theory. This *Hartree-Fock* Kernel creates a non-Hermitian matrix $\mathbf{\Omega}$. Please not that this calculation mode is as yet only available for single processor runs due to the lack of a non-Hermitian parallel eigenvalue solver.

Available Kernels and *libxc*

As of now, there is only the `pw-lda` f_{xc} kernel available for the TDDFT calculation in aims. If the user wants to make use of additional f_{xc} kernels he is requested to install *libxc*. *libxc* is a library of exchange-correlation functionals for density-functional theory (see [151]) available free under the Mozilla Public license from the internet. [2] Additionally, for the full TDDFT calculation it is possible to choose functionals at will that are available from this *libxc*. It is not required to have the DFT level calculation that generates the input energies ϵ for 3.75 using the same XC functional as the TDDFT calculation. You should really know what you are doing when you choose to experiment with different functionals and always keep in mind that the results might be unpredictable and not necessarily have any physical meaning at all. Nevertheless it might come in handy to be able to mix different hybrid functionals or do a DFT ground state calculation with a functional that does not provide and f_{xc} and still be able to do a TDDFT calculation on top of that when switching to functionals that do provide an f_{xc}.

Tags for general section of `control.in`

Tag: `neutral_excitation` (control.in)

Usage: `neutral_excitation type`
Purpose: Triggers the calculation of neutral excitations.
`type`: String that defines the type of calculation to be performed.

- tddft: Full TDDFT calculation
- tdhf: Full TDHF calculation (Kernel from 3.77, serial CPU only)
- rpa: random phase approximation only (set $f_{xc} = 0$ in 3.75)

With the keyword `neutral_excitation` the user can specify the calculation mode for the linear response theory.
Also the keyword `empty_states` should be set to 1000, or the keyword `calculate_all_eigenstates` should be used, to make sure the code generates all possible empty states provided from the basis set. This number will also be reduced automatically by the code to the maximum number that can be generated from the basis set.

Tag: `tddft_kernel` (control.in)

[2] http://www.tddft.org/programs/octopus/wiki/index.php/Libxc

Usage: tddft_kernel string
Purpose: Specify the origin of the TDDFT kernel for 3.75
string: pw-lda/pz-lda or libxc
Both pw-lda or pz-lda are built-in options in FHI-aims. They are equivalent to the keywords defined in xc .
When using libxc, one must specify the desired kernels through keywords tddft_x and tddft_c . Note that libxc is only possible when the user has compiled aims with libxc binding.

Tag: tddft_x (control.in)

Usage: tddft_x string
Purpose: Set the desired exchange kernel to use from libxc. The definition is from libxc's manual and can be found at the libxc[a] website.
string: The name of the selected exchange functional, i.e. XC_LDA_X

[a] http://www.tddft.org/programs/octopus/wiki/index.php/Libxc:manual

---

Tag: tddft_c (control.in)

Usage: tddft_c string
Purpose: Set the desired correlation kernel to use from libxc. The definition is from libxc's manual and can be found at the libxc website as well.
string: The name of the selected correlation functional, i.e. XC_LDA_C_PW

Tag: excited_mode (control.in)

Usage: excited_mode string
Purpose: Select which excitations will be calculated.
string: one of {singlet|triplet|both}. To calculate both singlets and triplets is set as the default when this keyword is omitted.

Tag: excited_states (control.in)

Usage: excited_states n
Purpose: Specify the number of excited state energies and oscillator strengths to be printed.
n: Integer number $n \in \mathbb{N},\ n \geq 0$

With the keyword excited_states the user can specify the number of excited states that will be printed in the output. The default for this is 50 if there are that many. Normal production runs will generally have many more (depending on the basis set and the number of electrons involved) that can easily reach beyond 10000. So to avoid a really huge output from this routine this defaut is set rather low. Feel free to choose any number of your liking, if it should be too large it will automatically be defaulting to all excited states. At the end of a calculation the file TDDFT_LR_Spectrum_(singlet/triplet).dat will be written into the directory the FHI-aims program was run in. It contains all

computed excitation energies and the corresponding oscillator strengths.

Tag: `casida_reduce_matrix` (control.in)

Usage: `casida_reduce_matrix boolean`
Purpose: Set to .true. if you want to reduce the energy range for the Kohn-Sham eigenvalues to be included in the computation.
`boolean`: .true. or .false.

Tag: `casida_reduce_occ` (control.in)

Usage: `casida_reduce_occ x`
Purpose: Specify the energy in Hartree below which the occupied states are cut off.
`x`: Cutoff energy in Hartree

Tag: `casida_reduce_unocc` (control.in)

Usage: `casida_reduce_unocc x`
Purpose: Specify the energy in Hartree above which the virtual states are cut off.
`x`: Cutoff energy in Hartree

3.27 Real-Time TDDFT

This functionality was tested thoroughly, but there is a possibility for remaining bugs or inconsistencies. If you have any questions regarding usage and/or functionality, if you encouter problems/bugs, or if you have suggestions - please contact joscha.hekele@uni-due.de or peter.kratzer@uni-due.de. Please cite our preliminary arXiv publication [99] when you publish results obtained with this functionality.

The current implementation incorporates a large set of control keywords of which only the most important are noted here. Several keywords controlling debugging/development features are not captured here but can be found in the code. Please contact us if you want to learn more about this.

Currently, one can do RT-TDDFT simulations for finite and periodic systems, including Ehrenfest dynamics for both cases. All modes of parallelism are supported.

Theory

Basic Principles

Real-Time TDDFT - as the name suggests - calculates the response of a Kohn-Sham system in real time. The time-dependent Kohn-Sham equation

$$i\frac{\partial}{\partial t}\psi^{\mathrm{KS}}(\mathbf{r},t) = \mathcal{H}^{\mathrm{KS}}[\rho(t),t]\psi^{\mathrm{KS}}(\mathbf{r},t) \tag{3.78}$$

is solved and describes the time-evolution of the electronic wave function $\psi^{\mathrm{KS}}(\mathbf{r},t)$. The Hamiltonian has an explicit time-dependence in this case, e.g. by an external electric field and/or a time-dependent external ionic potential.
In contrast to the linear response approach, real-time TDDFT is able to capture the whole nonlinear characteristic of the given system in real time. Possible applications include absorption spectrum calculations based on the dynamical dipole response, high-harmonic generation simulations or ion bombardment simulations based on non-adiabatically coupled electron-ion dynamics (Ehrenfest dynamics). Please see [163] for a comprehensive review.

Real-Time Propagation in FHI-aims

What is actually done is the time propagation of the atomic basis function coefficients $\{c_{in}(t)\}$ from the LCAO ansatz

$$\psi_n^{\mathrm{KS}}(\mathbf{r},t) = \sum_i^{N_{\mathrm{basis}}} c_{in}(t)\phi_i(\mathbf{r}-\mathbf{R}_{I(i)}) \quad , \quad n \in N^{\mathrm{occ}}.$$

The coefficients are expressed as a matrix by considering all (occupied) states and basis indices: $\mathbf{C} \in \mathbb{C}^{N_{\mathrm{basis}}\times N_{\mathrm{occ}}}$. The time-dependent KS matrix equation to be solved is then

$$\frac{d}{dt}\mathbf{C}(t) = -i\mathbf{S}^{-1}\mathbf{H}(t)\mathbf{C}(t)$$

where $\mathbf{S} \leftrightarrow \langle\phi_i|\phi_j\rangle$ is the overlap matrix and $\mathbf{H} \leftrightarrow \langle\phi_i|\mathcal{H}^{\mathrm{KS}}|\phi_j\rangle$ is the Hamiltonian matrix. The efficient and accurate solution of this equation is the key mechanic in a real-time TDDFT implementation. Note that very large basis sets will lead to an ill-conditioned overlap matrix, causing $\mathbf{S}^{-1}$ to be a problematic object, yielding entirely wrong results or blow-up.
The time-dependence of the Hamiltonian is implicit via the time-dependent density and explicit by the possible dependence on an external field, e.g. a laser or a dynamical ionic potential. An external electric field can here be incorporated via

$$\begin{aligned}\text{Length gauge}: &\quad H_{ij}(t) = H_{ij}^{\mathrm{KS}}[\rho(t)] + \mathbf{E}(t)\cdot\langle\phi_i|\mathbf{r}|\phi_j\rangle \\ \text{Velocity gauge}: &\quad H_{ij}(t) = H_{ij}^{\mathrm{KS}}[\rho(t)] - i\mathbf{A}(t)\cdot\langle\phi_i|\nabla|\phi_j\rangle + \frac{1}{2}\mathbf{A}^2(t)S_{ij}\end{aligned}$$

where the dipole approximation – neglecting any spatial dependence of the electric field – is used. Electric field $\mathbf{E}$ and vector potential $\mathbf{A}$ are connected by the relation $\mathbf{E} = -\partial_t\mathbf{A}$. Both gauges are physically equivalent but have different technical implications. In this implementation, only the velocity gauge can be applied in case of periodic systems. We note here that the (atomic ZORA) relativistic kinetic operator is currently not considered in the expression for the velocity gauge but that is on the to-do list (it can nevertheless be used). Scaled ZORA can not be applied.

Ehrenfest Dynamics

In Ehrenfest dynamics, non-adiabatically coupled motion between the electronic and ionic subsystems is simulated. Mobile atoms and thus mobile basis functions lead to a modified differential equation for the electrons,

$$\begin{aligned}\frac{d}{dt}\mathbf{C}(t) &= -i\mathbf{S}^{-1}(\mathbf{H}+\mathbf{G})\mathbf{C}(t), \\ G_{ij} &= i\langle\phi_i|\dot{\mathbf{R}}_{I(j)}\cdot\nabla|\phi_j\rangle,\end{aligned}$$

where the matrix $\mathbf{G}$ describes the time-dependence of the atomic basis functions and effectively conserves the norm of the time-propagated wavefunctions.
Furthermore, the forces on the nuclei are complemented by specific non-adiabatic contributions,

$$\mathbf{F}_I = \mathbf{F}_I^{\mathrm{HF}} + \mathbf{F}_I^{\mathrm{MP}} + \mathbf{F}_I^{\mathbf{XC}} + \mathbf{F}_I^{\mathrm{DBC}} + \mathbf{F}_I^{\mathrm{NC}},$$

where $\mathbf{F}_I^{\mathrm{HF}}$ are the standard Hellmann-Feynman forces, $\mathbf{F}_I^{\mathrm{MP}}$ are multipole force contributions and $\mathbf{F}_I^{\mathrm{XC}}$ are XC related forces, e.g. GGA. The force term

$$\mathbf{F}_I^{\mathrm{DBC}} = -\sum_n^{N_{\mathrm{occ}}}\sum_{ij}^{N_{\mathrm{basis}}} f_n c_{in}^* c_{jn}\Big[\langle\nabla_I\phi_i|\mathcal{H}^{\mathrm{KS}}|\phi_j\rangle + \langle\phi_i|\mathcal{H}^{\mathrm{KS}}|\nabla_I\phi_j\rangle\Big]$$

stands for "Dynamical Basis Correction" and can be seen as the time-dependent analogue to Pulay forces. The last term is denoted as the nonadiabatic coupling force, possibly consisting of two terms:

$$\mathbf{F}_I^{\mathrm{NC}} = \mathbf{F}_I^{\mathrm{EC}} + \mathbf{F}_I^{\mathbf{MC}}.$$

Including the full above-mentioned expression guarantees full energy- and momentum conservation, but usually only incorporating $\mathbf{F}_I^{\mathrm{EC}}$ is sufficient and noticeably cheaper. The energy-conserving force term can be written as

$$\begin{aligned}\mathbf{F}_I^{\mathrm{EC}} &= \sum_n^{N_{\mathrm{occ}}} f_n \mathbf{c}_n^\dagger \Big[\mathbf{H}\mathbf{S}^{-1}\mathbf{B}_I + \left(\mathbf{H}\mathbf{S}^{-1}\mathbf{B}_I\right)^\dagger\Big]\mathbf{c}_n,\\ \mathbf{B}_{ij,I} &= \langle\phi_i|\nabla_I|\phi_j\rangle.\end{aligned}$$

The remaining force term restoring momentum conservation is given as

$$\begin{aligned}\mathbf{F}_I^{\mathrm{MC}} &= i\sum_n^{N_{\mathrm{occ}}} f_n \mathbf{c}_n^\dagger \Big[\mathbf{W}_I^\dagger - \mathbf{W}_I + \mathbf{D}^\dagger\mathbf{S}^{-1}\mathbf{B}_I - \left(\mathbf{D}^\dagger\mathbf{S}^{-1}\mathbf{B}_I\right)^\dagger\Big]\mathbf{c}_n,\\ \mathbf{W}_{ij,I} &= \dot{\mathbf{R}}_I\langle\nabla_I\phi_i|\nabla_I\phi_j\rangle,\\ D_{ij,I} &= \dot{\mathbf{R}}_I\cdot\langle\phi_i|\nabla_I|\phi_j\rangle.\end{aligned}$$

The integration of the nuclear equations of motion is performed via a Velocity Verlet based algorithm, i.e.,

$$\begin{aligned}\dot{\mathbf{R}}_I(t+\Delta t/2) &= \dot{\mathbf{R}}(t) + \frac{\Delta t}{2M_I}\mathbf{F}_I(t),\\ \mathbf{R}_I(t+\Delta t) &= \mathbf{R}_I(t) + \Delta t\dot{\mathbf{R}}_I(t+\Delta t/2),\end{aligned}$$

where Δt is the corresponding Ehrenfest time stepping. Based on each coordinate change, a geometry update is performed. Please see [144, 178] for further details.

Observables

Depending on what task has to be performed, different physical observables are of interest in a RT-TDDFT simulation. The computation of the most important ones is briefly explained here. These are also computed and sent to output by default.

- **Electronic energy**: The time-dependent electronic energy for the current time step is computed via

$$\begin{aligned}E_{\mathrm{el}}^{\mathrm{KS}}(t) = &\sum_{n=1}^{\mathrm{N_{occ}}} f_n\langle\psi_n(t)|\mathcal{H}^{\mathrm{KS}}(t)|\psi_n(t)\rangle - \int \mathrm{d}^3\mathbf{r}\,\rho(\mathbf{r},t)v_{\mathrm{xc}}^{\mathrm{ad}}(\mathbf{r})\\ &+ E_{\mathrm{xc}}^{\mathrm{ad}}[\rho](t) - \frac{1}{2}\int \mathrm{d}^3\mathbf{r}\,\rho(\mathbf{r},t)v_{\mathrm{H}}(\mathbf{r},t) + E_{\mathrm{nuc}}(t).\end{aligned}$$

  where the adiabatic approximation is used, i.e., the instantaneous density $\rho(\mathbf{r},t)$ is employed to evaluate the functionals (this is indicated by the *ad* superscript). The electronic energy is not necessarily a conserved quantity, e.g. when energy is absorbed from an external field.

- **Total electronic and nuclear kinetic energy**: In case when Ehrenfest dynamics is performed, the nuclear kinetic energy contribution to the total energy has to be

taken into account, i.e.

$$\begin{aligned} E_{\mathrm{tot}}(t) &= E_{\mathrm{el}}^{\mathrm{KS}}(t) + E_{\mathrm{nuc,kin}}(t) \\ &= E_{\mathrm{el}}^{\mathrm{KS}}(t) + \frac{1}{2}\sum_I M_I \dot{\mathbf{R}}_I^2(t). \end{aligned}$$

In the absence of external fields, this quantity must be conserved.

- **Electronic dipole moment**: Important quantities that can be computed from the dynamical electronic dipole response are the polarisability $\alpha(\omega)$ and absorption strength $S(\omega)$:

$$\boldsymbol{\mu}(t) = \int d^3r\, \mathbf{r}\rho(t) \quad \rightarrow \quad \alpha_{ij}(\omega) = \frac{\mathcal{F}[\mu_i](\omega)}{\mathcal{F}[E_j](\omega)} \quad \rightarrow \quad S(\omega) = \frac{2\omega}{3\pi}\mathrm{Im}\Big\{\mathrm{Tr}\big[\alpha(\omega)\big]\Big\}$$

  The electronic dipole moment is by default computed for cluster systems. One can also compute tranisition dipole moments which are obtained by projection onto the ground state Kohn-Sham orbitals [36].

- **Electronic current**: From the current density $\mathbf{j}(\mathbf{r},t)$, one can compute the conductivity $\sigma(\omega)$ and the dielectric function $\epsilon(\omega)$ via the total current $\mathbf{I}(t)$:

$$\mathbf{j}(\mathbf{r},t) = \frac{1}{2}\sum_{nij} f_n\Big[c_{in}^*(t)c_{jn}(t)\langle\phi_i|-i\nabla+\mathbf{A}(t)|\phi_j\rangle + \mathrm{c.c.}\Big] \;\rightarrow\; \mathbf{I}(t) = -\frac{1}{V}\int d^3r\, \mathbf{j}(\mathbf{r},t)$$

$$\rightarrow\; \sigma_{ij}(\omega) = \frac{\mathcal{F}[I_i](\omega)}{\mathcal{F}[E_j](\omega)} \quad \rightarrow \quad \epsilon(\omega) = 1 + i4\pi\frac{\sigma(\omega)}{\omega}$$

- **Trajectories**: In case of Ehrenfest dynamics, trajectories are written to .xyz file/s.

Please note that a python-based post-processing tool named `eval_tddft.py` is located in the `utilities/rt-tddft` folder which is made for RT-TDDFT output files. It can currently compute, visualize and write the absorption strength function/polarisability (by input electronic dipole and external field) and also the conductivity/dielectric function (by input electronic current and external field) with possibly more functionality to come.

Important Notes

Some general important remarks regarding different topics are given here. Please also pay attention to this.

- **Exchange-Correlation**: Due to technical and/or formal limitations, not all available XC functionals can be used. We tested for LDA- and GGA-based functionals which should work fine. Other choices were not tested or are in development process and the code will stop for unreasonable settings. Hybrid functionals like B3LYP can be used for finite systems. This also applies for the long-range corrected LC-ωPBE functional.

- **Compatibility**: We tested our software for Intel-based parallelization together with the MKL numerical linear algebra library, and for GNU OpenMPI-based parallelization together with the numerical linear algebra libraries ScaLAPACK/OpenBLAS (versions 2019 / 2020). It seems like there exists a bug in the recent MKL versions (Intel Parallel Studio XE 2019 / 2020) that affects the linear equation solver subroutines `pzgetrs` and `pzgesv`, causing memory leakage at every call, resulting in excessive memory usage by the code. This was not observed with the non-Intel framework. These operations are conducted when the Crank-Nicolson propagator is used or when the scaling and squaring method for matrix exponentials is employed. We implemented a fix that will result in the code using explicit inversion instead of solving a linear system, i.e.:

$$\textbf{Linear solver}: \quad \text{solve} \quad \mathbf{AX} = \mathbf{B} \quad \text{for} \quad \mathbf{X} \tag{3.79}$$
$$\textbf{Inversion}: \quad \text{compute } \mathbf{A}^{-1} \quad \text{for} \quad \mathbf{X} = \mathbf{A}^{-1}\mathbf{B} \tag{3.80}$$

  For this, the user has to provide the flag

```
RT_TDDFT_crank_nicolson_solve_inv .true.
```

  in the `control.in` file.

Tags for general section of `control.in`

Core Settings

Tag: `RT_TDDFT_input_units` (control.in)

Usage: `RT_TDDFT_input_units units`

`units`: String that specifies in which unit system the input is given. If `atomic`, the remaining parameters are read in atomic units, if `spec`, the remaining parameters are read in as what we call spectroscopic (see table below).

This setting affects every physical input quantity given such that one has to pay attention to the consistent use of units. The following table depicts possible unit sets. Quite

Quantity	`atomic`	`spec`	Example
Time	$\hbar/E_h$	fs	Sim. time, steppings, pulse center/width
Frequency	$E_h/\hbar$	fs^{-1}	Ext. field frequency
Electric field	$E_h/a_0 e$	eV/Å	–

'typical' values for time steppings are 0.1 a.u. $= 0.0024$ fs or for electric field amplitudes 0.01 a.u. $= 51.42$ eV/Å (in the perturbative regime).
This keyword must be set by the user or the code will not run which is intended to prevent unit confusions.

Tag: `RT_TDDFT_propagation` (control.in)

Usage: `RT_TDDFT_propagation t_tot dt_prop dt_output`

Purpose: Performing a real-time TDDFT simulation.

`t_tot`: Total simulation time for which the time-propagation will be performed.

`dt_prop`: Time step which will be used.

`dt_output`: Time step for which output like energy, dipole, etc. is sent to standard output and is written to file/s.

The choice of the time step heavily affects the accuracy and stability of the time-propagation and is restricted by internal or external degrees of freedom, e.g., the spectral range of the Hamiltonian or the frequency of an external field oscillation. Generally, a small time step yields better results but significantly increases the computation time. The possible time step also depends on the propagation scheme which is explained later. A conservative choice is `dt_prop` ≤ 0.1 a.u. $(= 0.0024$ fs$)$ but 0.2 a.u. to 0.5 a.u. might also be possible. Remember using consistent units set by the `RT_TDDFT_input_units` keyword. Note that the time step is also a convergence parameter for spectral information.

Tag: `RT_TDDFT_td_field_gauge` (control.in)

Usage: `RT_TDDFT_td_field_gauge gauge`

Purpose: Setting the gauge for a possible external field (in the dipole approximation).

`gauge`: String. The field can be applied either in `length` gauge (electric field) or `velocity` gauge (vector potential). Default: `length` for cluster systems, `velocity` for periodic systems.

While this makes no difference formally, it has some technical implications. When doing periodic simulations, only the velocity gauge can be used. In the cluster case, the length gauge should be used because it is computationally cheaper. However, the formal gauge invariance is broken in any numerical approximation and it is likely possible that basis set and/or real-space grid settings converge differently for length and velocity gauge. Note that the gauge is a universal setting for each individual RT-TDDFT simulation and applies to all fields specified.

Tag: `RT_TDDFT_td_field` (control.in)

Usage: RT_TDDFT_td_field t_start t_end type freq cycle center width Ex Ey Ez

Purpose: Setting the type and shape of a possible external field (in the dipole approximation).

t_start: Start time after which the field is evaluated.

t_end: End time before which the field is evaluated (0 = infinite).

type: Integer that defines the field type as in rt_tddft_external_field.f90. Possible choices are defined below.

freq: Frequency if applicable (see type).

cycle: Cycle if applicable (see type).

center: Center if applicable (see type).

width: Width if applicable (see type).

Ex, Ey, Ez: Electric field amplitudes.

The choice of the external field depends on the type of calculation that one wants to perform. To obtain an absorption spectrum via the dynamical electronic dipole response, a delta-kick field can be applied by, e.g., setting type $=$ 2, center, width and Ez to some values which will apply a Gaussian pulse of given amplitude at center time/width and along the z-axis. width should then be chosen quite small, e.g., to 0.05 fs. Alternatively, one can provide a constant field for the first time step, which saves computation time and provides a sharp pulse naturally. This would be type $=$ 1 and t_end $=$ dt.
To apply a field pulse with a defined wavelength, one can, e.g., set type $=$ 3, freq, cycle, Ez which will result in a modulated Gaussian pulse of given wavelength around center time.
Multiple instances of this keyword with different settings can be given, e.g., to apply multiple different pulses over time. Internally, all defined fields are simply added up, so pay attention to temporal localization (if intended).
Note that the field amplitudes are always given as the amplitudes of the electric field, even in the case when the velocity gauge is used (see RT_TDDFT_td_field_gauge).
Currently, the following field types are implemented:

- 0: No field at all. This is the default when RT_TDDFT_td_field is not specified.
- 1: Constant field.
- 2: Delta kick pulse via Gaussian:

$$\mathbf{E}(t) = \mathbf{E}_0 \exp\left(\frac{t-t_0}{t_w}\right)\left(1+\exp\left(\frac{t-t_0}{t_w}\right)\right)^{-2}$$
$$\mathbf{A}(t) = -\mathbf{E}_0 t_w \left(1-\left(\exp\left(\frac{t-t_0}{t_w}\right)\right)^{-1}\right)$$

- 3: Localized field oscillation via sine-modulated Gaussian:

$$\mathbf{E}(t) = \mathbf{E}_0 \exp\left(-\frac{(t-t_0)^2}{2w^2}\right) \sin\Big(\omega(t-t_0)\Big)$$
$$\mathbf{A}(t) := \mathbf{E}(t)$$

  with $w = t_w/(2\sqrt{2\log(2)})$, $t_w = \mathrm{FWHM}$.

- 4: Sinusoidal field with reference frequency:

$$\mathbf{E}(t) = \mathbf{E}_0 \sin\left(\frac{\omega}{c}(t-t_0)\right)$$
$$\mathbf{A}(t) = \mathbf{E}_0 \frac{c}{\omega} \cos\left(\frac{\omega}{c}(t-t_0)\right)$$

- 5: Sin^2 envelope pulse:

$$\mathbf{E}(t) = \mathbf{E}_0 \Bigg(\sin\left(\frac{\pi}{c}(t-t_0)\right)^2 \sin(\omega(t-t_0)) - \left(\frac{\pi}{c\omega} \sin\left(\frac{2\pi}{c}(t-t_0)\right) \cos(\omega(t-t_0))\right) \Bigg)$$
$$\mathbf{A}(t) := \frac{\mathbf{E}_0}{\omega} \sin\left(\frac{\pi}{c}(t-t_0)\right)^2 \cos(\omega(t-t_0))$$

- 6: Ramp field:

$$\mathbf{E}(t) = (t-t_0)\mathbf{E}_0$$
$$\mathbf{A}(t) := \mathbf{E}(t)$$

where $t_0 =$ `t_start`, $\omega = 2\pi f$ and $f =$ `freq`, $c =$ `cycle`, $t_c =$ `center`, $t_w =$ `width`, $\mathbf{E}_0 = (E_x, E_y, E_z)^T$. Parameters not occurring in a function definition are not referenced and can be set to any value.

Note that user-defined fields can easily be included in the corresponding subroutine `src/rt-tddft/src/rt_tddft_ext_field.f90`.
Remember using consistent units set by the `RT_TDDFT_input_units` keyword.

Tag: `RT_td_field_spin_set` (control.in)

Usage: `RT_td_field_spin_set spin`

Purpose: Setting this will result in the external field only being applied to `spin` = 1 or 2 eigenvectors.

`spin`: Integer that specifies the spin component.

If this setting is specified for a collinear calculation (see keyword `spin` in the FHI-aims manual), only the spin = `spin` time-dependent eigenvectors will be excited by the specific external field. This may, e.g., be used to model triplet excitations.

Tag: `RT_TDDFT_propagator` (control.in)

Usage: `RT_TDDFT_propagator propagator`

Purpose: Setting the propagation scheme used in RT-TDDFT.

`propagator`: String that defines the numerical integration scheme. See below for possible options. Default: `crank_nicolson`.

The possible integration schemes are (see, e.g., [83, 41, 12]) :

- `exponential_midpoint` (EM):

$$\mathbf{U}(t_k + \Delta t, t_k) = \exp\left(-i\Delta t \mathbf{S}^{-1}\mathbf{H}(t_k + \Delta t/2)\right)$$

- `crank_nicolson` (CN): The order $\mathrm{N_{CN}}$ can be set via the keyword `RT_TDDFT_crank_nicolson_order order`.

$$\mathbf{U}(t_k + \Delta t, t_k) = \frac{\sum_{m=0}^{\mathrm{N_{CN}}}\left(-i\Delta t \mathbf{S}^{-1}\mathbf{H}(t_k + \Delta t/2)\right)^m}{\sum_{m=0}^{\mathrm{N_{CN}}}\left(i\Delta t \mathbf{S}^{-1}\mathbf{H}(t_k + \Delta t/2)\right)^m}$$

- `etrs` (Enforced Time-Reversal Symmetry):

$$\mathbf{U}(t_k + \Delta t, t_k) = \exp\left(-i\frac{\Delta t}{2}\mathbf{S}^{-1}\mathbf{H}(t_k + \Delta t)\right)\exp\left(-i\frac{\Delta t}{2}\mathbf{S}^{-1}\mathbf{H}(t_k)\right)$$

- `cfm4` (Commutator-Free Magnus Expansion 4):

$$\mathbf{U}(t_k + \Delta t, t_k) = \exp\left(-i\Delta t \mathbf{S}^{-1}\left(\alpha_1\mathbf{H}_1 + \alpha_2\mathbf{H}_2\right)\right)\exp\left(-i\Delta t \mathbf{S}^{-1}\left(\alpha_1\mathbf{H}_2 + \alpha_1\mathbf{H}_2\right)\right)$$
$$\alpha_{1/2} = \frac{1}{4} \mp \frac{\sqrt{3}}{6} \quad , \quad \mathbf{H}_{1/2} = \mathbf{H}\left(t_k + \left(\frac{1}{2} \mp \frac{\sqrt{3}}{6}\right)\Delta t\right)$$

- `m4` (Magnus Expansion 4):

$$\mathbf{U}(t_k + \Delta t, t_k) = \left(\frac{\Delta t}{2}\mathbf{S}^{-1}\left(\mathbf{H}_1 + \mathbf{H}_2\right) + \Delta t^2\frac{\sqrt{3}}{12}\left[\mathbf{S}^{-1}\mathbf{H}_2, \mathbf{S}^{-1}\mathbf{H}_1\right]\right)$$
$$\mathbf{H}_{1/2} = \mathbf{H}\left(t_k + \left(\frac{1}{2} \mp \frac{\sqrt{3}}{6}\right)\Delta t\right)$$

- `runge_kutta_4`: Standard explicit Runge-Kutta 4 scheme. Works only for very small time steps but is a good choice for doing accurate reference simulations because its properties are well-known.

- parallel_transport_cn: Crank-Nicolson integration scheme within the parallel transport approach [122]. This requires the accelerated Anderson solver, see RT_TDDFT_propagator_solver . [EXPERIMENTAL]

$$\begin{aligned}\mathbf{C}(t_k+\Delta t) = \; & \mathbf{C}(t_k) - i\frac{\Delta t}{2}\Big(\mathbf{S}^{-1}\mathbf{H}(t_k)\mathbf{C}(t_k) - \mathbf{C}(t_k)\Big(\mathbf{C}^{\dagger}(t_k)\mathbf{H}(t_k)\mathbf{C}(t_k)\Big)\\ & - i\frac{\Delta t}{2}\Big(\mathbf{S}^{-1}\mathbf{H}(t_k+\Delta t)\mathbf{C}(t_k+\Delta t)\\ & \qquad - \mathbf{C}(t_k+\Delta t)\Big(\mathbf{C}^{\dagger}(t_k+\Delta t)\mathbf{H}(t_k+\Delta t)\mathbf{C}(t_k+\Delta t)\Big)\end{aligned}$$

Note that all above mentioned implicit schemes are defined as implicit, as they require evaluation of the Hamiltonian matrix at some future time. Currently, there are two ways implemented on how to solve this type of equation which can be set by the RT_TDDFT_propagator_solver keyword.
It should also be noted here that, when using a predictor-corrector scheme (which is the default solver for all implicit propagators), accuracy can be enhanced by using extrapolation, see the keyword RT_TDDFT_extrapolate_predictor .

Some additional schemes are implemented for testing purposes, namely the M6 (m6), CFM4:3 (cfm4_3) and CFM6:5 (cfm6_5) schemes [12].

For general purpose, we recommend to use the crank_nicolson propagation scheme.

Tag: RT_TDDFT_propagator_solver (control.in)

Usage: RT_TDDFT_propagator_solver solver

Purpose: Lets one choose the solver technique used for time propagation and for the selected propagator.

solver: String that can be

- predictor_corrector: Standard semi-implicit predictor-corrector scheme.
- forward_extra: Uses extrapolation to approximate future Hamiltonian in implicit schemes. This is not an implicit scheme, as no corrector step is performed.
- anderson: Anderson acceleration to solve implicit schemes, see [236]. Currently only works with the crank_nicolson and parallel_transport_cn propagators. [EXPERIMENTAL]

This keyword must usually not be set explicitly since every RT_TDDFT_propagator has its own default setting for the solver. Anyway, one can set the solver manually, e.g., to use one of the implicit exponential propagators with extrapolation, which is much cheaper since the corrector integration of the Hamiltonian matrix is not applied (however, this may be much less stable, not justifying the setting). Some combinations are not possible and a warning will be given that default settings will be used. Please

use this option with caution.

Tag: `RT_TDDFT_use_precor_tol` (control.in)

Usage: `RT_TDDFT_use_precor_tol tol`

Purpose: Sets a tolerance of density change after which no more corrector steps are applied when multiple corrector steps may be performed.

`tol`: Float that sets convergence criterion to exit the corrector-loop. Not used by default.

Performing additional corrector steps until a convergence criterion is reached will formally increase accuracy. It should nevertheless be noted that the biggest error is probably introduced already by the predictor step and that significant improvements by doing additional corrector steps are not to be expected. Every corrector update involves another integration of the Hamiltonian matrix which is the most expensive part in RT-TDDFT usually. Before doing more corrector steps, it probably makes more sense to choose a smaller time step – if accuracy is a problem. If one wants to use this functionality anyway, values of 10^{-5}-10^{-9} would make sense usually. Note that a maximum number of 10 corrector steps is hardcoded to avoid unnecessarily many corrector updates (one should check settings when this happens).

Tag: `RT_TDDFT_precor_steps` (control.in)

Usage: `RT_TDDFT_precor_steps steps`

Purpose: Lets the user choose the (fixed) number of corrector steps to be done in the predictor-corrector solver.

`steps`: Integer setting a fixed number of corrector steps. Default: 1

Please see `RT_TDDFT_use_precor_tol` for further remarks. The default value is usually a good choice.

Tag: `RT_TDDFT_extrapolate_predictor` (control.in)

Usage: `RT_TDDFT_extrapolate_predictor extrapol`

Purpose: Extrapolation is used for the predictor step in case a predictor-corrector scheme is used.

`extrapol`: Bool that defines whether the Hamiltonian in a predictor propagator is extrapolated. Default: `.false.`

The Hamiltonian (matrix) used in the predictor step will be extrapolated by a given method which can be set via the `RT_TDDFT_ham_extrapolation` keyword. This also means that additional storage is needed. The extrapolated Hamiltonian (matrix) is applied in the predictor step. Tests indicate that extrapolation can noticeably improve accuracy. However, one should check this setting when unexpected instabilities occur.

Tag: `RT_TDDFT_ham_extrapolation` (control.in)

Usage: `RT_TDDFT_ham_extrapolation method order`

Purpose: Controls order and method of extrapolation for the Hamiltonian matrix if applicable.

`method`: String that can be either `linear` or `lagrange`. Default: `linear`.

`order`: Integer that defines the order of the extrapolation, i.e., how many past Hamiltonian matrices are used (and saved) for extrapolation. Default: 1.

The default setting indicating linear extrapolation works well in tests, whereas, in contrast, the Lagrange extrapolation approach sometimes produces odd results and should be used only with caution. Some of our tests indicate that higher-order extrapolation (≥ 4) can yield instabilities, thus one should check this.

Tag: `RT_TDDFT_propagator_predictor` (control.in)

Usage: `RT_TDDFT_propagator_predictor prop`

Purpose: A specific predictor propagator can be set in case a predictor-corrector scheme is used.

`prop`: String that defines the propagator to be used in the predictor step.

This only makes sense if an explicit scheme like `runge_kutta_4` is used as a predictor, or, and this is the usual setting here, if any applicable implicit scheme is used in combination with extrapolation. A well-functioning scheme consists of an extrapolated `exponential_midpoint` predictor in combination with the `cfm4` propagator for the corrector. This keyword is ignored if no predictor-corrector scheme is in use.

Tag: `RT_TDDFT_exponential_method` (control.in)

Usage: `RT_TDDFT_exponential_method method`

Purpose: Lets one choose the method for the calculation of matrix exponentials as required by some propagators.

`method`: String that can be either `eigenvectors`, `scaling_squaring` or `taylor`. Default: `eigenvectors`.

Currently, three different approaches to compute matrix exponentials needed for the exponential-type propagators can be used. The diagonalization-based approach is usually working perfectly but in the current implementation, it cannot be used for non-Hermitian matrices – this can be the case for effective Hamiltonian matrices occurring in Ehrenfest dynamics. In this case, the scaling and squaring - method [?] will be applied. Anyway, this setting will then be adjusted automatically. One can modify this setting mainly for benchmarking or testing. Note that using the `taylor` method requires to choose an order of the expansion, see

RT_TDDFT_exponential_taylor_order .

Tag: `RT_TDDFT_exponential_taylor_order` (control.in)

Usage: RT_TDDFT_exponential_taylor_order order

Purpose: Sets the order of the truncated Taylor expansion method for the matrix exponential.

`order`: Integer that sets the order. Default: 10.

The Taylor method applies to general matrices and can be an economic choice. However, the order must be set with caution, as we found that the here employed all-electron approach likely requires higher orders. It may not even work when heavy elements are involved. This setting also highly depends on the time step. We found values of 10-20 to work well in certain cases.

Tag: `RT_TDDFT_crank_nicolson_order` (control.in)

Usage: RT_TDDFT_crank_nicolson_order order

Purpose: Sets the order of the expansion used in the Crank-Nicolson propagator.

`order`: Integer that sets the order. Default: 1.

Usually, 1st order seems to suffice for most applications. However, it can be beneficial to use higher order, e.g., 2/3, when instabilities are observed, especially possibly for heavier elements.

Tag: `RT_TDDFT_crank_nicolson_solve_inv` (control.in)

Usage: RT_TDDFT_crank_nicolson_solve_inv solve

Purpose: Sets the inversion method in the Crank-Nicolson scheme.

`solve`: Boolean that sets the direct inversion method. Default: `.false.`

This only makes sense when the software bug explained above occurs. Setting the flag to `.true.` should resolve the issue. It is possible that the performance is impacted by this setting.

Tag: `RT_TDDFT_check_total_energy` (control.in)

Usage: RT_TDDFT_check_total_energy start_time thresh

Purpose: Compares the total energy to a fixed value at t = `start_time` and checks whether it is conserved below `thresh`.

`start_time`: Float that sets the time for which the reference energy is saved.

`thresh`: Float that sets the maximum energy deviation (in eV).

This can be used to prevent a simulation of 'blowing up', i.e., becoming unstable, as measured by the degree of energy conservation. The start time should be chosen such that the reference value makes sense: for example, when applying a finite resonant excitation, the total energy will increase, but should stay constant thereafter; thus, the reference value should be set a while after the excitation ends.

Tag: `RT_TDDFT_imaginary_time` (control.in)

Usage: `RT_TDDFT_imaginary_time flag`

Purpose: Peforms the simulation with imaginary time.

`flag`: Boolean that sets imaginary time. Default: `.false.`

Imaginary time propagation can be used as an alternative approach to the standard SCF procedure to converge a ground-state solution [?]. This does only require a time step as control parameter and may be used to converge situations where SCF fails. Note that we recommend the Crank-Nicolson propagator for this functionality (higher orders may also be beneficial in order to enable higher time steps).
After convergence is reached, the eigenvectors are written back into the native data structures so that postprocessing methods should be usable.

Tag: `RT_TDDFT_anderson_order_iter` (control.in)

Usage: `RT_TDDFT_anderson_order_iter order iter tol`

Purpose: Sets specific parameters that are important in the `anderson` solver.

`order`: Order of the scheme, i.e., the number of eigenvectors used for the Anderson acceleration. Default: 4.

`iter`: Maximum number of steps in the fixed-point iteration. Default: 30.

`tol`: Convergence criterion for the fixed-point iteration. Default: 1e-14.

These parameters are explained in detail by Walker et al. [236]. We refer to algorithm 'AA' in this reference. The parameter $m =$ `order`, $k_{\max} =$ `iter` as in the reference. The tolerance parameter depends on the setting `RT_TDDFT_anderson_cond_type` and should generally be chosen as a very small number. The most important parameter here is `iter`, which can be crucial to obtain a converged fixed-point iteration, but may also increase the possible numerical demand.

Tag: `RT_TDDFT_anderson_cond_type` (control.in)

Usage: `RT_TDDFT_anderson_cond_type type`

Purpose: Sets the type of the convergence check in the accelerated Anderson fixed-point iteration.

`type`: String that can be either `ev_maximum`, `ev_individual` or `delta_rho`. Default: `delta_rho`.

This defines which quantity is used as convergence measure in the fixed-point iteration. `ev_maximum` defines that $\max\{|\psi_n^{(i)} - \psi_n^{(i+1)}| \; \forall n \in \mathrm{N_{occ}}\}$ must be below `tol` as defined via `RT_TDDFT_anderson_order_iter`. `ev_individual` means that all solutions must be below the threshold `tol`. `delta_rho` only requires that the average change in charge density $\Delta\rho^i$ must be below the tolerance.

Ehrenfest Dynamics Settings

Tag: `RT_TDDFT_ehrenfest` (control.in)

Usage: `RT_TDDFT_ehrenfest type dt_geo`

Purpose: Performing a RT-TDDFT+Ehrenfest simulation, i.e., coupled electron-ion dynamics.

`type`: String that defines the Ehrenfest scheme. Must currently only be chosen as `default`.

`dt_geo`: Time step for geometry update and force computation.

The time step for Ehrenfest dynamics must obviously be at least the same value as for electron propagation. Usually, it can be chosen around 5- to 20-fold while maintaining stable propagation. Since updating the geometry and calculating forces is expensive, the Ehrenfest time step should be as large as possible. Additionally, the specific interplay of time propagation, force update and geometry update works very good when `dt_geo` $= N\times$ `dt` with $N = 2^n$. A reasonable choice for doing Ehrenfest dynamics should be `dt` $= 0.1$ a.u. in combination with `dt_geo` $= 0.4-0.64$ a.u. Monitoring energy conservation and norm conservation of the eigenvectors is advised when trying out settings.
Remember using consistent units set by the `RT_TDDFT_input_units` keyword.

Tag: `RT_TDDFT_ehrenfest_start_time` (control.in)

Usage: `RT_TDDFT_ehrenfest_start_time t_start`

Purpose: Setting the start time for Ehrenfest dynamics.

`t_start`: Time after which non-adiabatic forces are calculated and geometry is updated in case an Ehrenfest simulation is requested. Default: 0.

Usually, Ehrenfest dynamics start with the begin of a RT-TDDFT simulation when requested, but employing this key can modify the start time in case specific tasks require this.
Remember using consistent units set by the `RT_TDDFT_input_units` keyword.

Tag: `RT_TDDFT_ehrenfest_full_nc_forces` (control.in)

Usage: RT_TDDFT_ehrenfest_full_nc_forces key

Purpose: When requested, the full non-adiabatic Ehrenfest forces are evaluated.

key: Boolean that controls whether the full force contributions are calculated at each force computation step. Default: .false.

Usually, it seems that using the 'normal' non-adiabatic Ehrenfest forces conserving the energy (but not necessarily the momentum) is sufficient, but by using this keyword, the complete expression is evaluated – this might be important in specific situations and is of course more general. Anyway it is more expensive.

Tag: RT_TDDFT_ehrenfest_remove_com (control.in)

Usage: RT_TDDFT_ehrenfest_remove_com key

Purpose: Controls whether the center of mass is set to origin in Ehrenfest dynamics.

key: Boolean. Default: .false.

Output Settings

Tag: RT_TDDFT_write_file_prefix (control.in)

Usage: RT_TDDFT_write_file_prefix prefix

Purpose: Controls naming of all output files produced by the RT-TDDFT module.

prefix: String that is used to specify output files. Default: output.

All output files have the format PREFIX.rt-tddft.OBSERVABLE.SUFFIX or PREFIX.rt-tddft-ehrenfest.OBSERVABLE.SUFFIX where PREFIX can be set by the user to specify simulation settings. For example, energies are by default written to a file named output.rt-tddft.energies.dat. The corresponding quantities are always specified (format, units, etc.) in comment headers inside the files. Note that for any file output, existing files will not be overwritten. Instead, new files named PREFIX.N.rt-tddft.OBSERVABLE.SUFFIX with N $= 1, 2, ..$ will be created.

Tag: RT_TDDFT_write_ext_field (control.in)

Usage: RT_TDDFT_write_ext_field key

Purpose: Controls whether the time-dependent external field is written to a file.

key: Boolean. Default: .true.

The external field and all corresponding parameters are written to a file before a real-

time simulation in this case. This information can, e.g., be used for postprocessing or visualization. Note that our postprocessing tool `eval_tddft.py` requires this file.

Tag: `RT_TDDFT_write_cube` (control.in)

Usage: `RT_TDDFT_write_cube dt_cube`

Purpose: Controls whether time-dependent cube files are written.

`dt_cube`: Real value determining the time step for which output will be generated. If possible, this value should be much higher than the real-time time stepping since computing cubes is costly.

Default cube output is deactivated until this flag is given and the output frequency is specified. All options that can be used for cube output via the `output cube` keyword can be applied here and must be set in addition to this keyword.
Remember using consistent units set by the `RT_TDDFT_input_units` keyword.

Tag: `RT_TDDFT_output_level` (control.in)

Usage: `RT_TDDFT_output_level level`

Purpose: Controls the amount of console output written by the RT-TDDFT subroutine.

`level`: Integer with valid values of:

- 0: Min output - nearly nothing except basic information is written (NOTE: no information about physical observables is written in this case, too – use file output then)
- 1: Low output - some additional information is written, e.g., energies, dipole moment
- 2: Mid output - more is written out, e.g., convergence or accuracy parameters, timings
- 3: Max output - this prints info on anything that is being performed

Default: 1.

Since RT-TDDFT simulations are usually performed for a significant number of time-integration/density update/force calculation/etc. operations, the associated output will result in very large amounts of mostly unnecessary data which can be avoided by setting this keyword. The default should yield sufficient information on what is happening, but setting it to 2 will give some more information about numerical properties which can be important to follow.
Please note that important observables, e.g., energies, dipole moment, geometry, etc. are written to separate files which can be controlled further and which eases post-processing (see `RT_TDDFT_write_file_prefix`).

Tag: `RT_TDDFT_output_priority_override` (control.in)

Usage: `RT_TDDFT_output_priority_override priority`

Purpose: Lets one override the FHI-aims native output level for functionality used within RT-TDDFT.

`priority`: Integer that sets the output level. Default: 3 (minimal).

In the RT-TDDFT module, several routines from the parent code are called which may produce their own output. By default, the output level for these calls is set to minimal, i.e., not output will be sent to the console. However, this can be overridden by this keyword. We refer to the keywork `output_level` in the FHI-aims manual for specific choices.

Tag: `RT_TDDFT_output_energies` (control.in)

Usage: `RT_TDDFT_output_energies key_std key_file`

Purpose: Controls whether time-dependent energies are calculated, sent to output and written to a file.

`key_std`: Boolean controlling whether energies are written to standard output. Default: `.true.`

`key_file`: Boolean controlling whether energies are written to a file. Default: `.false.`

Energy should always be observed in order to validate a properly running simulation, i.e., total energy conservation. Note that when values are written to a file too, the corresponding file is named `PREFIX.rt-tddft.energies.dat` where PREFIX can be changed via the `RT_TDDFT_write_file_prefix` keyword.

Tag: `RT_TDDFT_output_dipole` (control.in)

Usage: `RT_TDDFT_output_dipole key_std key_file`

Purpose: Controls whether the time-dependent dipole moment is calculated, sent to output and written to a file.

`key_std`: Boolean controlling whether the dipole moment is written to standard output. Default: `.true.` for cluster systems, else `.false.`

`key_file`: Boolean controlling whether dipole moment is written to a file. Default: `.false.`

The electronic dipole is usually an observable of major interest in many RT-TDDFT applications. Note that when values are written to a file too, the corresponding file is named `PREFIX.rt-tddft.dipole.dat` where PREFIX can be changed via the `RT_TDDFT_write_file_prefix` keyword.

Tag: `RT_TDDFT_output_dipole_xyz` (control.in)

Usage: `RT_TDDFT_output_dipole_xyz out_x out_y out_z`

Purpose: Controls whether the x, y, z components of the electronic dipole moment are actually computed.

`out_x`: Boolean controlling whether the x-component is computed. Default: `.true.`

`out_y`: Boolean controlling whether the y-component is computed. Default: `.true.`

`out_z`: Boolean controlling whether the z-component is computed. Default: `.true.`

This keyword can be used to save unnecessary computation time for components that are not of interest. However, the computational effort for observables is generally low, as compared to the remaining functionality.

Tag: `RT_TDDFT_output_transition_dipole` (control.in)

Usage: `RT_TDDFT_output_transition_dipole key_std key_file state_min state_max`

Purpose: Controls whether and which transition dipole moments are computed.

`key_std`: Boolean controlling whether transition dipole moments are written to standard output. Default: `.false.`

`key_file`: Boolean controlling whether transition dipole moments are written to files. Note that a file for each transition is created. Default: `.false.`

`state_min`: Index of the lowest occupied state that will be considered.

`state_max`: Index of the highest unoccupied state that will be considered.

The computation of transition dipole moments can be requested here. The lower and upper index define the set of occupied/unoccupied Kohn-Sham states for which time-dependent transition dipole moments will be computed. It is necessary that the native keyword `empty_states` is set accordingly high (e.g., to the number of basis functions or, e.g., 10000), else it is possible that not enough unoccupied states are available in the corresponding arrays. This is one of the rather computationally expensive observables.

Tag: `RT_TDDFT_output_transition_dipole_xyz` (control.in)

Usage: RT_TDDFT_output_transition_dipole_xyz out_x out_y out_z

Purpose: Controls whether the x, y, z components of the transition dipole moments are actually computed.

out_x: Boolean controlling whether the x-component is computed. Default: .true.

out_y: Boolean controlling whether the y-component is computed. Default: .true.

out_z: Boolean controlling whether the z-component is computed. Default: .true.

This keyword can be used to save unnecessary computation time for components that are not of interest.

Tag: RT_TDDFT_output_state_dipoles (control.in)

Usage: RT_TDDFT_output_state_dipoles key_std

Purpose: Controls whether time-dependent dipole moments of individual occupied states are calculated and sent to standard output.

key_std: Boolean. Default: .false.

If given, the dipole expectation values for all occupied (and thus, time-propagated) states are computed and sent to output. Note that this computation requires the explicit evaluation of the dipole matrix $\langle\phi_i|\mathbf{r}|\phi_j\rangle$ and could cause additional computational cost.

Tag: RT_TDDFT_output_magnetic_moment (control.in)

Usage: RT_TDDFT_output_magnetic_moment key_std key_file

Purpose: Controls whether the time-dependent magnetic moment is calculated, sent to output and written to a file.

key_std: Boolean controlling whether the magnetic moment is written to standard output. Default: .false.

key_file: Boolean controlling whether the magnetic moment is written to a file. Default: .false.

The magnetic moment is usually an observable of major interest in RT-TDDFT applications for circular dichroism. Note that when values are written to a file too, the corresponding file is named
PREFIX.rt-tddft.magmom.dat where PREFIX can be changed via the
RT_TDDFT_write_file_prefix keyword.

Tag: RT_TDDFT_output_magnetic_moment_xyz (control.in)

Usage: RT_TDDFT_output_dipole_xyz out_x out_y out_z

Purpose: Controls whether the x, y, z components of the magnetic moment are actually computed.

out_x: Boolean controlling whether the x-component is computed. Default: .true.

out_y: Boolean controlling whether the y-component is computed. Default: .true.

out_z: Boolean controlling whether the z-component is computed. Default: .true.

This keyword can be used to save unnecessary computation time for components that are not of interest. However, the computational effort for observables is generally low, as compared to the remaining functionality.

Tag: RT_TDDFT_output_current (control.in)

Usage: RT_TDDFT_output_current key_std key_file

Purpose: Controls whether the time-dependent electronic current is calculated, sent to output and written to a file.

key_std: Boolean controlling whether the current is written to standard output. Default: .false. for cluster systems, else .true.

key_file: Boolean controlling whether the current is written to a file. Default: .false.

The electronic current is often the main observable of interest in periodic RT-TDDFT simulations (analog to the dipole in finite systems). Note that when values are written to a file too, the corresponding file is named PREFIX.rt-tddft.current.dat where PREFIX can be changed via the RT_TDDFT_write_file_prefix keyword.

Tag: RT_TDDFT_ehrenfest_output_trajectory (control.in)

Usage: RT_TDDFT_ehrenfest_output_trajectory key_std key_file

Purpose: Controls whether time-dependent coordinates, velocities and forces are sent to output and written to a file in Ehrenfest dynamics.

key_std: Boolean controlling whether trajectory information is written to standard output. Default: .true. for Ehrenfest dynamics, else .false.

key_file: Boolean controlling whether trajectory information is written to a file. Default: .false.

Note that when values written are to a file too, the corresponding file is named PREFIX.rt-tddft-ehrenfest.trajectory.xyz where PREFIX can be changed via the RT_TDDFT_write_file_prefix keyword.

Tag: `RT_TDDFT_output_n_excited` (control.in)

Usage: `RT_TDDFT_output_n_excited key_std`

Purpose: Controls whether the time-dependent number of excited electrons based on ground-state projection is computed and sent to standard output.

`key_std`: Boolean controlling whether this quantity is computed and written to standard output. Default: `.false.`

This quantity is computed as

$$N_e^{\mathrm{ex}}(t) = \sum_{n=1}^{\mathrm{N_{occ}}} f_n(N_{electrons} - |\langle\psi_n(0)|\psi_n(t)\rangle|^2), \tag{3.81}$$

i.e., via projection of time-dependent orbitals on ground-state orbitals [?]. This refers to the number of electrons that are possibly excited within the introduced dynamics.

Restart Settings

Tag: `RT_TDDFT_restart_write` (control.in)

Usage: `RT_TDDFT_restart_write t_write_restart`

Purpose: Writes a RT-TDDFT restart file for a specified time.

`t_write_restart`: Real value corresponding to specific time for which a restart file should be written.

All necessary information to perform a RT-TDDFT restart is written to file/s in this case. In detail, all the eigencoefficients, coordinates, settings, etc. Multiple entries of this keyword for different times can be given. Note that this can potentially produce huge files, especially when periodic boundary conditions are in use (as for every $\mathbf{k}$-point, eigenvectors are written).

Tag: `RT_TDDFT_restart_write_period` (control.in)

Usage: `RT_TDDFT_restart_write_period t_write_restart_period`

Purpose: Writes a RT-TDDFT restart file repeatedly after each specified time period.

`t_write_restart`: Real value corresponding to specific time period after which a restart file should be written.

See `RT_TDDFT_restart_write` for details. When doing very demanding simulations and/or very long simulations, one should set a sensible value here to avoid needing to restart from the beginning after a job was canceled. Writing a restart file automatically when a kill signal is received is on the to-do list.

Tag: `RT_TDDFT_restart_read` (control.in)

Usage: `RT_TDDFT_restart_read restart_file`

Purpose: Reads a RT-TDDFT restart file and starts the simulation based on it.

`restart_file`: String describing the corresponding restart file.

Besides the physical quantities, the restart file contains all relevant real-time simulation parameters which are compared to what is given in control.in. The settings must be the same or the simulation will abort. This also works for Ehrenfest dynamics. Note that when performing a restart, FHI-aims will run a SCF cycle, but the resulting eigenvectors, etc. will be overwritten from what is read in the restart file - based on this, the real-time simulation will be re-initialized.

Tags for `geometry.in`

Tag: `RT_TDDFT_initial_velocity` (geometry.in)

Usage: `RT_TDDFT_initial_velocity v_x v_y v_z`

Purpose: Initial velocity of corresponding (i.e. last specified) `atom` when peforming RT-TDDFT-Ehrenfest dynamics.

`v_x, v_y, v_z`: Initial velocities in units of Å/ps.

Putting initial kinetic energy into the ionic subsystem is a common initial condition used in Ehrenfest dynamics, e.g. when simulating ion bombardment or to evaluate molecular potential energy surfaces.

3.28 NEO-DFT

This functionality was tested to our best knowledge, but there is a possibility for remaining bugs or inconsistencies. If you have any questions regarding usage and/or functionality, if you encouter problems/bugs, or if you have suggestions - please contact jxu@unc.edu or ryzhou@live.unc.edu. Please cite our publication [239] when you publish results obtained with this functionality.

The current implementation incorporates a large set of control keywords of which only the most important are noted here. Several keywords controlling debugging/development features are not captured here but can be found in the code (e.g. neo/src/neo_variable.f90). Please contact us if you want to learn more about this.

Currently, one can do NEO-DFT simulations for finite and periodic systems. All modes of parallelism are supported.

Theory

Multicomponent DFT via NEO method for Kohn-Sham system

The NEO formalism in the framework of multicomponent DFT leads to a set of coupled electron-proton Kohn-Sham (KS) equations,

$$\hat{H}^e_{\mathbf{k}}\psi^e_{i,\mathbf{k}}(\boldsymbol{r}^e) = \left[-\frac{1}{2}\nabla^2 + v^e_{\text{eff}}(\boldsymbol{r}^e)\right]\psi^e_{i,\mathbf{k}} = \epsilon^e_{i,\mathbf{k}}\psi^e_{i,\mathbf{k}}(\boldsymbol{r}^e), \tag{3.82}$$

$$\hat{H}^p\psi^p_i(\boldsymbol{r}^p) = \left[-\frac{1}{2M^p}\nabla^2 + v^p_{\text{eff}}(\boldsymbol{r}^p)\right]\psi^p_i = \epsilon^p_i\psi^p_i(\boldsymbol{r}^p), \tag{3.83}$$

where M^p is the proton mass, $\psi^e_{i,\mathbf{k}}$ and $\epsilon^e_{i,\mathbf{k}}$ are the KS orbitals and eigenvalues of the electrons, and ψ^p_i and ϵ^p_i are the KS orbitals and eigenvalues of the protons. The effective potentials are defined as

$$v^e_{\text{eff}}(\boldsymbol{r}^e) = v_{\text{ext}}(\boldsymbol{r}^e) + v^e_{\text{es}}(\boldsymbol{r}^e) - v^p_{\text{es}}(\boldsymbol{r}^e) + \frac{\delta E^e_{\text{xc}}[\rho^e]}{\delta\rho^e} + \frac{\delta E_{\text{epc}}[\rho^e,\rho^p]}{\delta\rho^e}, \tag{3.84}$$

$$v^p_{\text{eff}}(\boldsymbol{r}^p) = -v_{\text{ext}}(\boldsymbol{r}^p) - v^e_{\text{es}}(\boldsymbol{r}^p) + v^p_{\text{es}}(\boldsymbol{r}^p) + \frac{\delta E^p_{\text{xc}}[\rho^p]}{\delta\rho^p} + \frac{\delta E_{\text{epc}}[\rho^e,\rho^p]}{\delta\rho^p}. \tag{3.85}$$

where v_{ext}, v^e_{es}, and v^p_{es} are electrostatic potentials for the classical nuclei, electrons, and quantum protons. E^e_{xc} and E^p_{xc} are the exchange-correlation energies of the electrons and the quantum protons, respectively. E_{epc} is the correlation energy between the electrons and quantum protons, which is given as a functional of both the electron density and the proton density in the multicomponent DFT formalism. In most cases, the KS wave functions for the quantum protons are extremely localized in real space such that the overlap of proton wave functions from different unit cells is close to zero. Thus, the Brillouin zone (BZ) integration can be neglected for the protons even for extended systems, whereas the electron KS wave functions require the BZ integration. With

Equations 3.82 and 3.83, the ground state of the multicomponent system of electrons and quantum protons needs to be solved self-consistently in the presence of the electrostatic potential from the classical nuclei. Further discussions can be found in our publication [239].

Tags for general section of `control.in`

Core Settings

Tag: `NEO_type` (control.in)

Usage: `NEO_type type`

Purpose: Performing a NEO-DFT simulation

`type`: Integer that specifies the type of NEO calculation to be performed.

- 0: Do not include NEO-DFT. (Default)
- 1: NEO-DFT.
- 2: RT-NEO-TDDFT.

The overall switch to turn on a NEO-DFT calculation in FHI-aims. Must set manually to enable NEO simulations.

Tag: `NEO_basis_type` (control.in)

Usage: `NEO_basis_type type`

`type`: Integer that specifies the type of basis used for the quantization of protons.

- 1: Primitive spherical gaussian basis whose exponent and weight are provided later by `NEO_basis` keyword for each individual protons. (Default)
- 2: Even-tempered spherical gaussian basis whose radial part is defined as $\phi_{i,l}(r) = C_i * r^l * e^{-\alpha*\beta^{i-1}*r^2}$, where i runs from 1 to N (`NEO_basis_n`), l runs from 0 to l_{max} (`NEO_basis_l_max`), C_i is the automatically generated normalize constant, α and β is defined by `NEO_basis_alpha` and `NEO_basis_beta` keyword.

Tag: `NEO_density_fitting_type` (control.in)

Usage: NEO_density_fitting_type type

type: Integer that specifies the type of auxiliary basis used for the Resolution of the Identity (RI) method.

- 0: Not using RI method. Instead, electrostatic potential are calculated with a similar approach as described in [28], and exchange are calculated with four center two electron integrals between selected proton pairs.
- 2: Even-tempered spherical gaussian basis whose radial part is defined as $\psi_{i,l}(r) = C_i * r^l * e^{-\alpha*\beta^{i-1}*r^2}$, where i runs from 1 to N (NEO_df_n), l runs from 0 to l_{max} (NEO_df_l_max), C_i is the automatically generated normalize constant, α and β is defined by NEO_df_alpha and NEO_df_beta keyword. (Default)

By default, RI are used for calculating both coulomb and exchange terms. One can optionally use different ways to calculate exchange terms. See NEO_RI_exchange_type keyword for more detail.

Tag: NEO_epc_type (control.in)

Usage: NEO_epc_type type

type: Integer that specifies the type of eletron proton correlation (EPC) functionals.

- 0: Not using EPC functional. (Default)
- 171: Using the epc17-1 functional [35].
- 17 (or 172): Using the epc17-2 functional [240].

It has been shown that EPC functionals have significant impact on the total energy of the calculations. Including EPC will result in a better estimation of zero-point energy caused by the quantization of protons.

Tag: NEO_initial_guess_type (control.in)

Usage: NEO_initial_guess_type type

type: Integer that specifies how the initial guess for proton state are determined.

- 1: Starting the simulation from solving the proton states where protons can no feel each other. This option works great with single quantized proton cases.
- 2: Starting the simulation from eigenvectors provided in file "neo_eigenvectors.in".
- 3: Starting the simulation from n random eigenvectors, where n is the number of quantized protons. (Default)

In some special cases, a bad initial guess can lead to the simulation end up with a local minimum, where protons are very delocalized. When that happened, one can either change the NEO_initial_guess_type option or reorder the proton basis provided in NEO_basis to get a different initial guess.

Tag: `NEO_RI_exchange_type` (control.in)

Usage: NEO_RI_exchange_type type

type: Integer that specifies the type of RI method used especially for calculating the exchange terms in the simulation.

- 1 : Using the RI-V method. (Should be avoid in extended systems)
- 3 : Using the RI-LVL method. (Default)
- 32: Using the RI-LVL method but ultilize a sparsed storage and algorithm for linear algebra. This option will be slower than option 3 for small systems but faster for large systems.
- 5 : Not using RI method at all. This should be only used when once use NEO_density_fitting_type 0.
- 51: Directly calculating exchange terms using the wavefunctions of protons, but use RI-V to calculate electrostatic potentials.
- 53: Directly calculating exchange terms using the wavefunctions of protons, but use RI-LVL to calculate electrostatic potentials.

One can compare the results using option 3 and option 53 to check the performance of the current choice of auxiliary basis for the RI method.

Tag: `NEO_occ_proton_type` (control.in)

Usage: NEO_occ_proton_type type

type: Integer that specifies how the proton occupation number is treated.

- 1: Ground state. One protron per state starting from the lowest energy. (Default)
- 2: Read from FILE neo_occ.in, where first line indicate the number of occupied states, and the second line gives the occupation numbers from low energy state to high energy state separated by space.

Proton self consistent field loop (SCF) settings

Tag: `NEO_scf_proton_max_step` (control.in)

Usage: NEO_scf_proton_max_step N

Purpose: Defines the maximium proton SCF steps performed before updating the proton contribution and going back to electron SCF loops, if the proton SCF never meets the convergence criteria.

N is a integer. Default: 500.

The default value is set to be large for safety. Generally, N around 20 should be enough for convergence.

Tag: NEO_scf_accuracy_rho (control.in)

Usage: NEO_scf_proton_max_rho value

Purpose: Convergence criterion of proton SCF, based on proton charge density.

value is a small positive real number, against which the volume-integrated root-mean-square of the proton charge density between the present and previous proton SCF is checked. Default: 1e-6.

This convergence criterion is the most important SCF convergence for proton SCF cycles, one many consider to set this depend on the total number of protons.

Tag: NEO_scf_accuracy_eev (control.in)

Usage: NEO_scf_proton_max_eev value

Purpose: Convergence criterion of proton SCF, based on the change of proton eigenvalues.

value is a small positive real number, against which the change of the sum of proton eigenvalue between the present and previous proton SCF is checked. Default: 1e-3.

Tag: NEO_scf_accuracy_dm (control.in)

Usage: NEO_scf_proton_max_step value

Purpose: Convergence criterion of proton SCF, based on the change of proton density matrix.

value is a small positive real number, against which the largest changed element () in proton density matrix between the present and previous proton SCF is checked. Default: 1e-3.

This convergence criterion can be redundant when proton basis has linear dependence.

Tag: NEO_scf_accuracy_err (control.in)

Usage: NEO_scf_proton_max_err value

Purpose: Convergence criterion of proton SCF, based on the change of the error matrix defined by the DIIS procedure.

value is a small positive real number, against which the largest changed element in the error matrix between the present and previous proton SCF is checked. Default: 1e-5.

Tag: NEO_scf_diis_type (control.in)

Usage: NEO_scf_diis_type type

type: Integer that specifies the type of DIIS method used to enhance the convergence of proton scf cycle.

- 0: Not using DIIS method. Use the calculated proton Hamiltonian without any mixing method.
- 1: Using a altered DIIS method to update proton Hamiltonian before solving the proton KS equation.
- 2: Similar to option 1, but using a orthonormal residual. This option has better performance when there are linear dependence in proton basis.
- 3: Using the original DIIS method to update proton Hamiltonian before solving the proton KS equation.
- 4: Similar to option 4, but using a orthonormal residual. This option has better performance when there are linear dependence in proton basis. (Default)

For the simlations with extremely small proton basis sets (e.g. only $1s$ orbital), the DIIS protpagtion may fail. In that case, one can run the simulation without DIIS method.

Tag: NEO_scf_diis_max_step (control.in)

Usage: NEO_scf_diis_max_step N

Purpose: Defines how many proton SCF steps are stored for the DIIS procedure. All steps are stored for the first N steps, and then replace the oldest with the newer one.

N is a integer. Default: 8.

Generally, the default value is sufficient for DIIS. One may increase N if the DIIS procedure performs badly.

Numerical cutoff values for speeding up

Tag: NEO_proton_hartree_rcut (control.in)

Usage: NEO_proton_hartree_rcut value

Purpose: Cutoff radius, within which the electrostatic potential of a quantum proton is evaluated and updated for SCF loops. For extended systems with Ewald method, all real space evaluations of electrostatic potential are limited in this radius as well.

value is a real number in Å. Default: 30 Bohr.

NEO_proton_hartree_rcut is the most important parameter in the NEO-DFT work flow and must be set. Based on tests, 15 to 20 Åis a reasonable choice for the balence of accuracy and performance.

Tag: NEO_proton_hartree_rclassic (control.in)

Usage: NEO_proton_hartree_rclassic value

Purpose: Cutoff radius, within which the electrostatic potential of a quantum proton is calculated analytically. Beyond the radius, the electrostatic potential is approximated by classical multipoles located at the quantum proton center.

value is a real number in Å. Default: -1.0, the radius cutoff for the multipole approximation is set automatically to ensure that beyond this radius the proton charge density is less than NEO_proton_hartree_charge_threshold_rclassic for all quantum protons.

This parameter should be less than NEO_proton_hartree_rcut . With the default setting, FHI-aims will print out NEO_proton_hartree_rclassic used in the calculation in its output. One can choose a larger number to ensure better numerical results.

Tag: NEO_basis_rcut (control.in)

Usage: NEO_basis_rcut value

Purpose: Cutoff radius, beyond which the value of proton wavefunctions are set as zero.

value is a real number in Å. Default: -1.0, the radius cutoff for proton basis is set automatically to ensure that beyond this radius its value is less than NEO_proton_basis_wf_threshold_rcut for all basis functions.

With the default setting, FHI-aims will print out NEO_basis_rcut used in the calculation in its output. One can choose a larger number to ensure better numerical results. Due to the localized nature of quantum proton, this value can be less than 5 Å.

Tag: NEO_exchange_rcut (control.in)

Usage: NEO_exchange_rcut value

Purpose: Cutoff radius, beyond which the exchange effect between the quantum proton pair is ignored.

value is a real number. Default: 4.

As discussed in our paper[239], the exchange term is mainly for counteract the self-interaction of quantum protons. So one can limited the exchange effect within a very small radius. It has been tested that, value can be set to very close to 0 if no ghost/empty protons are included. Otherwize, it is better to increase value to cover all ghost centers for a quantum proton.

Tag: NEO_rho_cut (control.in)

Usage: NEO_rho_cut value

Purpose: Cutoff value, less than which proton charge density is set as zero.

value is a small real number. Default: 1e-30.

Tag: NEO_proton_hartree_charge_threshold_rclassic (control.in)

Usage: NEO_proton_hartree_charge_threshold_rclassic value

Purpose: Parameter used to determine NEO_proton_hartree_rclassic by default. This will only be referenced if NEO_proton_hartree_rclassic is not set.

value is a small positive real number. Default: 1e-30.

Tag: NEO_proton_basis_wf_threshold_rcut (control.in)

Usage: NEO_proton_basis_wf_threshold_rcut value

Purpose: Parameter used to determine NEO_basis_rcut by default. This will only be referenced if NEO_basis_rcut is not set.

value is a small positive real number. Default: 1e-30.

Ewald method settings

Tag: NEO_use_ewald (control.in)

Usage: NEO_use_ewald flag

Purpose: Determins whether the Ewald method are used to calculate electrostatic potentials.

flag is a logical string, either .false. or .true.. Default: .true. for isolated systems, .false. for extended systems.

By default, for molecular cases, FHI-aims only update the electrostatic potential of a given quantum proton within a limited radius defined by NEO_proton_hartree_rcut . For periodical cases, the electrostatic potential on all grid points are updated with the Ewald method.

Tag: **NEO_ewald_l_max** (control.in)

Usage: NEO_ewald_l_max l_{max}

l_{max}: Integer that specifies the highest angular momentum used in the Ewald Summation. Higher angular momentums are treated locally within NEO_proton_hartree_rcut . Default: same value as NEO_df_l_max .

Proton basis and auxiliary basis for Resolution of the Identity

---

Tag: **NEO_basis_n** (control.in)

Usage: NEO_basis_n N

N: Integer that specifies the rank of the Even-tempered spherical gaussian basis. See NEO_basis_type keyword for more detail. Default: 8.

Tag: **NEO_basis_l_max** (control.in)

Usage: NEO_basis_l_max l_{max}

l_{max}: Integer that specifies the highest angular momentum of the Even-tempered spherical gaussian basis. See NEO_basis_type keyword for more detail. Default: 2.

Tag: **NEO_basis_alpha** (control.in)

Usage: NEO_basis_alpha alpha

alpha: Real value that specifies the inital exponent of the Even-tempered spherical gaussian basis. See NEO_basis_type keyword for more detail. Default: $2 * \sqrt{2}$.

Tag: **NEO_basis_beta** (control.in)

Usage: NEO_basis_beta beta

beta: Real value that specifies the interval of exponents of the Even-tempered spherical gaussian basis. See NEO_basis_type keyword for more detail. Default: $\sqrt{2}$.

Tag: **NEO_df_n** (control.in)

Usage: NEO_df_n N

N: Integer that specifies the rank of the Even-tempered spherical gaussian auxiliary basis. See NEO_df_type keyword for more detail. Default: 10.

Tag: NEO_df_l_max (control.in)

Usage: NEO_df_l_max l_{max}

l_{max}: Integer that specifies the highest angular momentum involved. See NEO_df_type keyword for more detail. Default: 3.

Tag: NEO_df_alpha (control.in)

Usage: NEO_df_alpha alpha

alpha: Real value that specifies the inital exponent of the Even-tempered spherical gaussian auxiliary basis. See NEO_df_type keyword for more detail. Default: $2 * \sqrt{2}$.

Tag: NEO_df_beta (control.in)

Usage: NEO_df_beta beta

beta: Real value that specifies the interval of exponents of the Even-tempered spherical gaussian auxiliary basis. See NEO_df_type keyword for more detail. Default: $\sqrt{2}$.

Tag: NEO_basis (control.in)

Usage: NEO_basis

Purpose: Defines the basis of quantum protons in NEO-DFT simulations.

Tag: NEO_basis_end (control.in)

Usage: NEO_basis_end

Purpose: End the defination of quantum proton basis in NEO-DFT simulations.

NEO_basis is use to invoke the defination of proton basis for individual protons, one can add NEO_basis_end to end the defination of basis set for safety. Examples to use these keywords are provided in "neo.in_example".

NEO_basis **sub-tag:** H (control.in)

Usage: H index pos_x pos_y pos_z
[l_1 alpha_1 weight_1]
[l_2 alpha_2 weight_2]
[...]
[l_N alpha_N weight_N]

Purpose: Adds a (ghost) quantum proton center in NEO-DFT calculation

index is a integer for the index of hydrogen atom (proton) declared by atom in geometry.in. Optionally, negtive number represents for adding a ghost center that do not have charge distributions, which usually works together with empty in geometry.in.
pos_x, pos_y, pos_z are optional parameters that redefine the position of proton center. By default, the position of proton center are read from geomerty.in if optionpos_x, pos_y, pos_z are not provided. Providing new position for proton centers are still in beta phase and not guaranteed to work.
l_i is a character, specifying the angular momentum (*s*, *p*, *d*, *f*, ...) of the ith gaussian type orbital.
alpha_i is a real number [in bohr^{-2}], specifying the exponents α_i of the ith gaussian type orbital.
weight_i is a real number, specifying the weight of the ith gaussian type orbital. For primitive gaussian basis, weight should be 1.0.

3.29 Bethe-Salpeter equation: BSE

The serial BSE runs both with and without Tamm-Dancoff Approximation (TDA) and print out both results by default. The parallel BSE runs only with TDA at the moment.

The BSE eigenvalue equation in matrix form is:

$$\begin{bmatrix} A & B \\ -B & -A \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \Lambda \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

where Λ: Excitation energies; X_1, X_2: Excitation eigenvectors; matrix element A and B are calculated by quasiparticle energies, Coulomb and screened Coulomb integrals;

$$A_{ia}^{jb} = -\alpha^{S/T} < ia|V|jb > + < ij|W(\omega = 0)|ab > + (E_a^{QP} - E_i^{QP})\delta_{ij}\delta_{ab}$$

$$B_{ia}^{jb} = -\alpha^{S/T} < ia|V|bj > + < ib|W(\omega = 0)|aj >$$

where i, j: occupied states; a, b: unoccupied states; $\alpha^{S/T} = 2$ for singlet states; 0 for triplet states. The TDA considers only block A, which is symmetric and easy to solve.

Tags for general section of `control.in`

Tag: `neutral_excitation` (control.in)

Usage: `neutral_excitation type`
Purpose: Triggers the calculation of neutral excitations.
`type`: String that defines the type of calculation to be performed.

- bse: Full BSE calculation without TDA for serial run; TDA BSE for parallel run.

Also the keyword `empty_states` should be set to a large number (e.g., 1000), or the keyword `calculate_all_eigenstates` should be used, to make sure the code generates all possible empty states provided from the basis set. This number will also be reduced automatically by the code to the maximum number that can be generated from the basis set.

Tag: `read_write_qpe` (control.in)

Usage: `read_write_qpe type`
Purpose: Specify write quasiparticle energies (qpe) in GW calculation or read qpe in BSE calculation
`type`: String that specify read or write qpe, or both.

- w: write qpe to a file, used together with `qpe_calc`
- r: read qpe from a file, if this value('r') is set, a file "energy_qp" should be provided by the user. The energy in the file "energy_qp" should be in hartree unit.
- wr or rw: write qpe in GW and read qpe in BSE

Tag: `bse_s_t` (control.in)

Usage: `bse_s_t type`
Purpose: Specify whether singlet or triplet excitation energies should be calculated in BSE calculation.
`type`: String that specify wheter singlet or triplet, can not do both at the moment.

- singlet: Singlet states calculated in BSE.
- triplet: Triplet states calculated in BSE.

3.30 CC-aims: Interface to CC4S

The Coupled Cluster 4 Solids (CC4S) code offers a continuously increasing variety of wavefunction-based beyond-DFT methods for both molecules and materials (https://manuals.cc4s.org/user-manual/index.html). In order to perform a correlated calculation, CC4S requires a set of quantities including at least the single-particle energies and the so-called Coulomb vertex[115].

The CC-aims library[168] (https://gitlab.com/moerman1/fhi-cc4s), which serves as an interface between FHI-aims and CC4S, performs the calculation and preparation of the Coulomb vertex and of other quantities including metadata in a format which CC4S can understand.

A basic guide on CC-aims for FHI-aims users including installation and a tutorial can be found in the CC-aims manual (https://moerman1.gitlab.io/fhi-cc4s/for_FHIaims_users/).

To run the CC-aims interface in the most basic way after a completed HF/KS-SCF-calculation, it sufficies to add `output cc4s` to the general section of the `control.in`.

Also note, that as CC-aims is used as a post-SCF feature, it respects the `frozen_core_postscf` -keyword. Hence, if you want to reduce the computational cost of your CC4S-calculation by neglecting core-like HF/KS-states, you can do this by specifying `frozen_core_postscf` .

However, FHI-aims offers an additional set of CC-aims-related keywords, which extend the functionality of the interface and which are listed below.

Tags for general section of `control.in`

Tag: `cc4s_debug` (control.in)

Usage: `cc4s_debug flag`

Purpose: This keyword signals to the CC-aims library that more verbose output should be generated. In additiion to that, CC4S-input files which would be written to a binary/unformatted file, like the Coulomb vertex, will be output as human-readable files. As some of the output quantities, including the Coulomb vertex, can become huge this debug-option is only recommended for small systems.

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

Tag: `cc4s_screen_thresh` (control.in)

Usage: cc4s_screen_thresh value

Purpose: If this keyword is specified, a principal component analysis (PCA) is performed during the construction of the Coulomb vertex, thus reducing the final size of that quantity. During the PCA, all singular values of the Coulomb vertex smaller than value are discarded.

value is a small positive real number.
Default: No PCA is performed.

Tag: **cc4s_screen_num** (control.in)

Usage: cc4s_screen_num number

Purpose: Similarly to cc4s_screen_thresh value this keyword invokes a PCA of the Coulomb vertex. In the case of cc4s_screen_num , however, number is the maximum absolute number of singular values kept after the PCA.

number is a positive integer.
Default: No PCA is performed.

To avoid confusion, please only specify either cc4s_screen_thresh or cc4s_screen_num but not both.

3.31 DFPT - density functional perturbation theory for lattice dynamics and homogeneous electric fields

All these DFPT features are written by Honghui Shang and coworkers at FHI, when using routines related to DFPT, please contact shang@fhi-berlin.mpg.de.

Please note that efforts are currently underway to unify the existing DFPT implementations within FHI-Aims.

The density functional perturbation theory (DFPT) is in principle the perturbation form of DFT, which is only needed for the second and higher order derivatives ($2n+1$ theorem). For example, for the calculation of vibrational frequencies and phonon band-structures (second order derivative) the response of the electronic structure to a nuclear displacement (first order derivative) is needed. These derivatives can be calculated in the framework of density-functional perturbation theory (DFPT). DFPT provide access to many fundamental physical phenomena, such as superconductivity, phonon-limited carrier lifetimes in electron transport and hot electron relaxation, Peierls instabilities, the renormalization of the electronic structure due to nuclear motion, Born effective charges, phonon-assisted transitions in spectroscopy as well as infrared and Raman spectra.

We support DFPT calculations

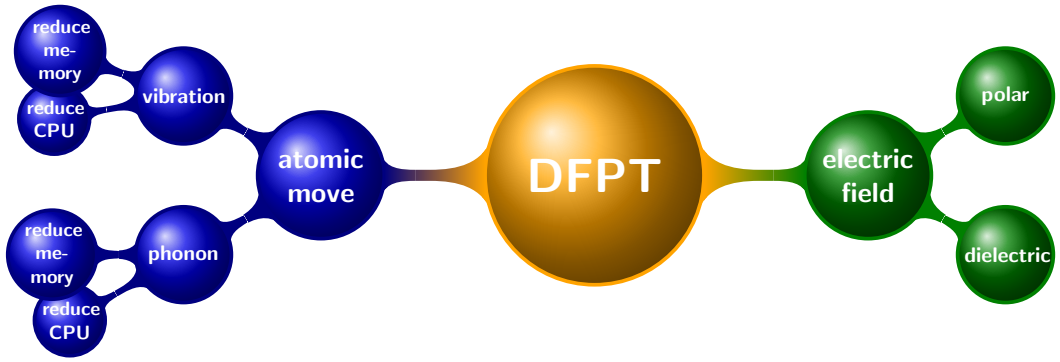
- For vibrations in non-periodic systems and phonons in periodic systems.
- For homogeneous electric fields in non-periodic systems (polarizability) and in periodic systems (dielectric constant).

There are three key references that provide the technical background for these sections:

1. **Lattice dynamics calculations based on density-functional perturbation theory in real space**
   Honghui Shang, Christian Carbogno, Patrick Rinke, Matthias Scheffler
   Comp. Phys. Comm.**215**, 26 (2017)

2. **The moving-grid effect in density functional calculations of harmonic vibration frequencies using numeric atom-centered grids**
   Honghui Shang, to be submitted to J. Chem. Phys.

3. **All-Electron, Real-Space Perturbation Theory for Homogeneous Electric Fields: Theory, Implementation, and Application within DFT**
   Honghui Shang, Nathaniel Raimbault, Patrick Rink, Matthias Scheffler, Mariana Rossi, Christian Carbogno
   New J. Phys. **20**, 073040 (2018)



atomic displacement

- For molecules:
  "DFPT vibration", "DFPT vibration_reduce_memory"
- For solid :
  "DFPT phonon", "DFPT phonon_reduce_memory"

electric field

- For molecules:
  "DFPT polarizability"
- For solid :
  "DFPT dielectric"

Figure 3.6: The DFPT feathers in FHI-aims.

Theory

Lattice Dynamics

In order to get vibration/phonon frequencies, first we need to get dynamical matrix, Let's define a lattice vector $\mathbf{R}_{Im}$ as

$$\mathbf{R}_{Im} = \mathbf{R}_I + \mathbf{R}_m, \tag{3.86}$$

whereby $\mathbf{R}_m$ denotes an arbitrary linear combination of $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$. And the dynamical matrix $D_{IJ}(\mathbf{q})$ is a Fourier transform of harmonic Hessian matrix (Force constant) $\Phi^{harm}_{Im,J}$.

$$\begin{aligned} D_{IJ}(\mathbf{q}) &= \frac{1}{\sqrt{M_I M_J}} \sum_m \Phi^{harm}_{Im,J} \exp\left(i\mathbf{q}\cdot\mathbf{R_m}\right) \\ &= \frac{1}{\sqrt{M_I M_J}} \sum_m \frac{d^2 E_{KS}}{d\mathbf{R}_{Im} d\mathbf{R}_J} \exp\left(i\mathbf{q}\cdot\mathbf{R_m}\right) \end{aligned} \tag{3.87}$$

Since the finite $(3N \times 3N)$ dynamical matrix $\mathbf{D}(\mathbf{q})$ would in principle have to be determined for an infinite number of $\mathbf{q}$-points in the Brillouin zone. Its diagonalization would produce a set of $3N$ $\mathbf{q}$-dependent eigenfrequencies $\omega_\lambda(\mathbf{q})$ and -vectors $\mathbf{e}_\lambda(\mathbf{q})$.

In order to derive the second order derivatives for total energy **analytically** (which is the key idea for the DFPT approach), the ground state total energy and force is derived first and we could get this second order derivatives directly. It is should be noted that, here real-space DFPT method is used, so that we could get the real-space force constant directly.

In FHI-aims, the total energy is calculated by using band-energy, a derivation for cluster systems is a following, and the extending to extended systems is straightforward.

$$\begin{aligned} E_{KS} = -\frac{1}{2}\sum_i < \phi_i|\nabla^2|\phi_i > - \int n(\mathbf{r}) \sum_I \frac{Z_I}{|\mathbf{r}-\mathbf{R}_I|} d\mathbf{r} + \\ \frac{1}{2}\int\int \frac{n(\mathbf{r})n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}d\mathbf{r}' + \frac{1}{2}\sum_I \sum_{J\neq I} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} + E_{xc}(n) \end{aligned} \tag{3.88}$$

using

$$\Rightarrow \hat{h_{ks}} = -\frac{1}{2}\nabla^2 - \sum_I \frac{Z_I}{|\mathbf{r}-\mathbf{R}_I|} + \int \frac{n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' + v_{xc}(\mathbf{r})$$

we have

$$\begin{aligned} &= \sum_i < \phi_i|\hat{h_{ks}}|\phi_i > - \int [n(\mathbf{r})v_{xc}(\mathbf{r})] d\mathbf{r} + E_{xc}(n) \\ &- \frac{1}{2}\int\int \frac{n(\mathbf{r})n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}d\mathbf{r}' + \frac{1}{2}\sum_I \sum_{J\neq I} \frac{Z_I Z_J}{|\mathbf{u}_I - \mathbf{u}_I|} \\ &= \sum_i f_i \epsilon_i - \int [n(\mathbf{r})v_{xc}(\mathbf{r})] d\mathbf{r} + E_{xc}(n) \end{aligned} \tag{3.89}$$

$$-\frac{1}{2}\int\int\frac{n\mathbf{r})n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}d\mathbf{r}'+\frac{1}{2}\sum_I\sum_{J\neq I}\frac{Z_IZ_J}{|\mathbf{R}_I-\mathbf{R}_I|} \tag{3.90}$$

$$=\sum_i f_i\epsilon_i-\int[n(\mathbf{r})v_{xc}(\mathbf{r})]d\mathbf{r}+E_{xc}(n)$$

$$-\frac{1}{2}\int\int\frac{n(\mathbf{r})n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}d\mathbf{r}'+\frac{1}{2}\int n(\mathbf{r})\sum_I\frac{Z_I}{|\mathbf{r}-\mathbf{R}_I|}d\mathbf{r}$$

$$+\frac{1}{2}\sum_I\sum_{J\neq I}\frac{Z_IZ_J}{|\mathbf{R}_I-\mathbf{R}_J|}-\frac{1}{2}\int n(\mathbf{r})\sum_I\frac{Z_I}{|\mathbf{r}-\mathbf{R}_I|}d\mathbf{r} \tag{3.91}$$

$$=\sum_i f_i\epsilon_i-\int[n(\mathbf{r})v_{xc}(\mathbf{r})]d\mathbf{r}+E_{xc}(n)$$

$$-\frac{1}{2}\int n(\mathbf{r})[\sum_I V_I^{free}(|\mathbf{r}-\mathbf{R}_I|)+\delta V_I(|\mathbf{r}-\mathbf{R}_I|)]d\mathbf{r}$$

$$-\frac{1}{2}\sum_I Z_I[\sum_J V_J^{free}(|\mathbf{R}_J-\mathbf{R}_I|)+\sum_{J\neq I}\delta V_J(|\mathbf{R}_J-\mathbf{R}_I|)] \tag{3.92}$$

$$=\sum_i f_i\epsilon_i-\int[n(\mathbf{r})v_{xc}(\mathbf{r})]d\mathbf{r}+E_{xc}(n)$$

$$-\frac{1}{2}\int n(\mathbf{r})[\sum_I V_I^{es,tot}(|\mathbf{r}-\mathbf{R}_I|)]d\mathbf{r}$$

$$-\frac{1}{2}\sum_I Z_I[V_I^{es,tot}(0)+\sum_{J\neq I}V_J^{es,tot}(|\mathbf{R}_J-\mathbf{R}_I|)] \tag{3.93}$$

Here Eq.(3.93) is exactly the one to calculate Kohn-Sham total energy. An extension expression to extended system is in Eq.3.94

$$E_{tot}=-\frac{1}{N_k}\sum_i^{uc}f_{\mathbf{k}i}\epsilon_{\mathbf{k}i}-\int_{uc}[n(\mathbf{r})v_{xc}(\mathbf{r})]d\mathbf{r}+E_{xc}(n) \tag{3.94}$$

$$-\frac{1}{2}\int_{uc}n(\mathbf{r})[\sum_{I,m}V_I^{es,tot}(|\mathbf{r}-\mathbf{R}_{I,m}|)]d\mathbf{r}$$

$$-\frac{1}{2}\sum_I Z_I[V_I^{es,tot}(0)+\sum_{(J,m)\neq(I,0)}V_J^{es,tot}(|\mathbf{R}_{Jm}-\mathbf{R}_I|)]$$

Then we get the analytical force expression for cluster systems:

$$F_I=-\frac{dE_{KS}}{d\mathbf{R}_I}=\mathbf{F}_I^{HF}+\mathbf{F}_I^P+\mathbf{F}_I^M$$

$$-[-\frac{\partial(\int n(\mathbf{r})\sum_I\frac{Z_I}{|\mathbf{r}-\mathbf{R}_I|}d\mathbf{r})}{\partial\mathbf{R}_I}+\frac{\partial(\frac{1}{2}\sum_I\sum_{J\neq I}\frac{Z_IZ_J}{|\mathbf{R}_I-\mathbf{R}_J|})}{\partial\mathbf{R}_I}]$$

$$-\sum_{\mu\nu}[P_{\mu\nu}\int(\frac{\partial\chi_\mu}{\partial\mathbf{R}_I}\hat{h}_{ks}\chi_\nu+\chi_\mu\hat{h}_{ks}\frac{\partial\chi_\nu}{\partial\mathbf{R}_I})d\mathbf{r}-W_{\mu\nu}\frac{\partial S_{\mu\nu}}{\partial\mathbf{R}_I}]$$

$$-\int [n(\mathbf{r})-n^{MP}(\mathbf{r})]\frac{\partial V^{MP}(\mathbf{r})}{\partial \mathbf{R}_I}d\mathbf{r} \tag{3.95}$$

Here $P_{\mu\nu}$ refers to density matrix and $W_{\mu\nu}$ refers to energy density matrix. Here the first term is Hellmann-Feynman force using Hellmann-Feynman theorem ; The second term is Pulay term, it comes from basis set dependence of atomic coordinate; The third term is multipole force, it count the contribution from multipole expansion error for Hartree energy calculation (multipole Poisson solver).

Under periodic boundary conditions, we get the following expression for force for extended systems:

$$\mathbf{F}_J^{HF}=Z_J[\frac{\partial V_J^{es,tot}(0)}{\partial \mathbf{R}_J}+\sum_{(I,m)\neq(J,0)}(\frac{\partial V_I^{es,tot}(|\mathbf{R}_J-\mathbf{R}_{Im})|}{\partial \mathbf{R}_J})] \tag{3.96}$$

$$\mathbf{F}_J^{P}=-\frac{2}{N_k}\sum_{\mathbf{k},i,\mu,\nu}\int f_{i\mathbf{k}}C^*_{\mu i\mathbf{k}}\frac{\partial \varphi^*_{\mu\mathbf{k}}(\mathbf{r})}{\partial \mathbf{R}_J}(\hat{h}_{ks}-\epsilon_{i\mathbf{k}})C_{\nu i\mathbf{k}}\varphi_{\nu\mathbf{k}}(\mathbf{r})d\mathbf{r}\ , \tag{3.97}$$

In our real-space DFPT method, this harmonic Hessian matrix is calculated directly, as explained in our paper. The Hessian can be split into two part

$$E^{(2)}_{KS}=\frac{d^2E_{KS}}{d\mathbf{R}_Id\mathbf{R}_J}=\Phi^{HF}_{I,J}+\Phi^{P}_{I,J}+\Phi^{MP}_{I,J} \tag{3.98}$$

$$=[-\frac{\partial^2(\int n(\mathbf{r})\sum_I\frac{Z_I}{|\mathbf{r}-\mathbf{R}_I|}d\mathbf{r})}{\partial \mathbf{R}_I\partial \mathbf{u}_J}+\frac{\partial^2(\frac{1}{2}\sum_I\sum_{J\neq I}\frac{Z_IZ_J}{|\mathbf{R}_I-\mathbf{R}_J|})}{\partial \mathbf{R}_I\partial \mathbf{R}_J}]$$

$$+\sum_{\mu\nu}\frac{\partial P_{\mu\nu}}{\partial \mathbf{R}_J}\int(\frac{\partial \chi_\mu}{\partial \mathbf{R}_I}\hat{h}_{ks}\chi_\nu+\chi_\mu\hat{h}_{ks}\frac{\partial \chi_\nu}{\partial \mathbf{R}_I})d\mathbf{r}$$

$$+\sum_{\mu\nu}P_{\mu\nu}\int[\frac{\partial^2\chi_\mu}{\partial \mathbf{R}_I\partial \mathbf{R}_J}\hat{h}_{ks}\chi_\nu+\frac{\partial \chi_\mu}{\partial \mathbf{R}_I}\frac{\partial \hat{h}_{ks}}{\partial \mathbf{R}_J}\chi_\nu+\frac{\partial \chi_\mu}{\partial \mathbf{R}_I}\hat{h}_{ks}\frac{\partial \chi_\nu}{\partial \mathbf{R}_J}]d\mathbf{r}$$

$$+\sum_{\mu\nu}P_{\mu\nu}\int[\frac{\partial \chi_\mu}{\partial \mathbf{R}_J}\hat{h}_{ks}\frac{\partial \chi_\nu}{\partial \mathbf{R}_I}+\chi_\mu\frac{\partial \hat{h}_{ks}}{\partial \mathbf{R}_J}\frac{\partial \chi_\nu}{\partial \mathbf{R}_I}+\chi_\nu\hat{h}_{ks}\frac{\partial^2\chi_\nu}{\partial \mathbf{R}_I\partial \mathbf{R}_J}]d\mathbf{r}$$

$$-(\sum_{\mu\nu}\frac{\partial E_{\mu\nu}}{\partial \mathbf{R}_J}\frac{\partial S_{\mu\nu}}{\partial \mathbf{R}_I}+\sum_{\mu\nu}E_{\mu\nu}\frac{\partial^2S_{\mu\nu}}{\partial \mathbf{R}_I\partial \mathbf{R}_J})$$

$$+\int\frac{\partial[n(\mathbf{r})-n^{MP}(\mathbf{r})]}{\partial \mathbf{R}_I}\frac{\partial V^{MP}(\mathbf{r})}{\partial \mathbf{R}_J}d\mathbf{r}$$

$$+\int[n(r)-n^{MP}(r)]\frac{\partial^2V^{MP}(r)}{\partial \mathbf{R}_I\partial \mathbf{R}_J}d\mathbf{r}$$

It can also be divided into three part: Hellmann-Feynman Hessian, Pulay Hessian, Multipole Hessian. In real calculation, we drop multipole Hessian due to its value is only 10^{-3} times small compared with the other two.

Under periodic boundary conditions, we get the following expression for force constants:

$$\Phi^{harm}_{Is,J} = \frac{d^2\mathbf{E}^{KS}}{d\mathbf{R}_{Is}d\mathbf{R}_J} = -\frac{d\mathbf{F}_J}{d\mathbf{R}_{Is}} = -\frac{d\mathbf{F}_{Is}}{d\mathbf{R}_J} = \Phi^{HF}_{Is,J} + \Phi^{P}_{Is,J} \ . \tag{3.99}$$

$$\begin{aligned} \Phi^{HF}_{Is,J} &= -Z_J \left(\frac{d}{d\mathbf{R}_J} \frac{\partial V^{es}_J(0)}{\partial \mathbf{R}_J} \right) \delta_{Is,J0} \\ &\quad -Z_J \left(\frac{d}{d\mathbf{R}_J} \frac{\partial V^{es,tot}_I(|\mathbf{R}_J - \mathbf{R}_{Is})|}{\partial \mathbf{R}_{Is}} \right) (1 - \delta_{Is,J0}) \ , \end{aligned} \tag{3.100}$$

in which $\delta_{Is,J0} = \delta_{IJ}\delta_{s0}$ denotes a multi-index Kronecker delta.

For the sake of readability, its total derivative is split into four terms:

$$\Phi^{P}_{Is,J} = \Phi^{P-P}_{Is,J} + \Phi^{P-H}_{Is,J} + \Phi^{P-W}_{Is,J} + \Phi^{P-S}_{Is,J} \ . \tag{3.101}$$

The first term

$$\Phi^{P-P}_{Is,J} = 2 \sum_{\mu m,\nu n} \left(\frac{dP_{\mu m,\nu n}}{d\mathbf{R}_J} \right) \int \frac{\partial \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is}} \hat{h}_{ks} \chi_{\nu n}(\mathbf{r}) \, d\mathbf{r} \tag{3.102}$$

accounts for the response of the density matrix $P^{m,n}_{\mu m,\nu n}$. The second term

$$\begin{aligned} \Phi^{P-H}_{Is,J} &= 2 \sum_{\mu m,\nu n} P_{\mu m,\nu n} \cdot & (3.103) \\ &\quad \left(\int \frac{\partial^2 \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is} \partial \mathbf{R}_J} \, \hat{h}_{ks} \, \chi_{\nu n}(\mathbf{r}) \, d\mathbf{r} \right. & (3.104) \\ &\quad + \int \frac{\partial \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is}} \frac{d\hat{h}_{ks}}{d\mathbf{R}_J} \chi_{\nu n}(\mathbf{r}) \, d\mathbf{r} & (3.105) \\ &\quad \left. + \int \frac{\partial \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is}} \hat{h}_{ks} \frac{\partial \chi_{\nu n}(\mathbf{r})}{\partial \mathbf{R}_J} \, d\mathbf{r} \right) & (3.106) \end{aligned}$$

accounts for the response of the Hamiltonian $\hat{h}_{ks}(\mathbf{k})$, while the third and fourth term

$$\Phi^{P-W}_{Is,J} = -2 \sum_{\mu m,\nu n} \frac{dW_{\mu m,\nu n}}{d\mathbf{R}_J} \int \frac{\partial \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is}} \chi_{\nu n}(\mathbf{r}) \, d\mathbf{r} \tag{3.107}$$

$$\Phi^{P-S}_{Is,J} = -2 \sum_{\mu m,\nu n} W_{\mu m,\nu n} \frac{\partial}{\partial \mathbf{R}_J} \int \frac{\partial \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is}} \chi_{\nu n}(\mathbf{r}) \, d\mathbf{r} \tag{3.108}$$

for the response of the energy weighted density matrix $W_{\mu m,\nu n}$ and the overlap matrix $S_{\mu m,\nu n}$, respectively, Please note that in all four contributions many terms vanish due to the fact that the localized atomic orbitals $\chi_{\mu m}(\mathbf{r})$ are associated with one specific atom/periodic image $\mathbf{R}_{J(\mu)m}$, which implies, e.g.,

$$\frac{\partial \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is}} = \frac{\partial \chi_{\mu m}(\mathbf{r})}{\partial \mathbf{R}_{Is}} \delta_{J(\mu)m,Is} \ . \tag{3.109}$$

In the above force constants, it is clear that the first order density matrix $\frac{\partial P_{\mu\nu}}{\partial \mathbf{R}_J}$ and the first order energy density matrix $\frac{\partial W_{\mu\nu}}{\partial \mathbf{R}_J}$ are needed. These first order qualtities are obtained in the DFPT cycle. The flowchart of our DFPT cycle for lattice dynamics is shown in Fig.3.7.

Our resulting frequencies look like this:

```
DFPT-Results:

List of all frequencies found:
  Mode number      Frequency [cm^(-1)]   IR-intensity [D^2/Ang^2]
            1           -2282.47061452                 0.00000000
            2           -2282.47061452                 0.00000000
            3              -0.00004008                 0.00000001
            4               0.00003217                 0.00000000
            5               0.00006390                 0.00000000
            6            5546.30603353                 0.00000000
```

For vibration calculation, there are there keywords to choose:

- DFPT vibration
- DFPT vibration_reduce_memory
- DFPT vibration_with_moving_grid_effect

A comparison for these three features as well as DFT calculation is shown in Fig.3.8

In phonon calculation, only Gamma point results is printed in the output like this:

```
 ============================================
 DFPT-Results: for q1=  0.000000000000000E+000  0.000000000000000E+000
  0.000000000000000E+000
 ============================================

 List of all frequencies found:
  Mode number      Frequency [cm^(-1)]
            1           -2376.08047043
            2           -2376.08047043
            3              -0.00004497
            4               0.00001694
            5               0.00009141
            6            5113.49725256
```

A typical phonon band structure is shown for Graphen in Fig. 3.9

A scaling test for DFPT code for lattice dynamics is has been done for Si system, with up to 1024 atoms in the unit cell see Fig.3.10.

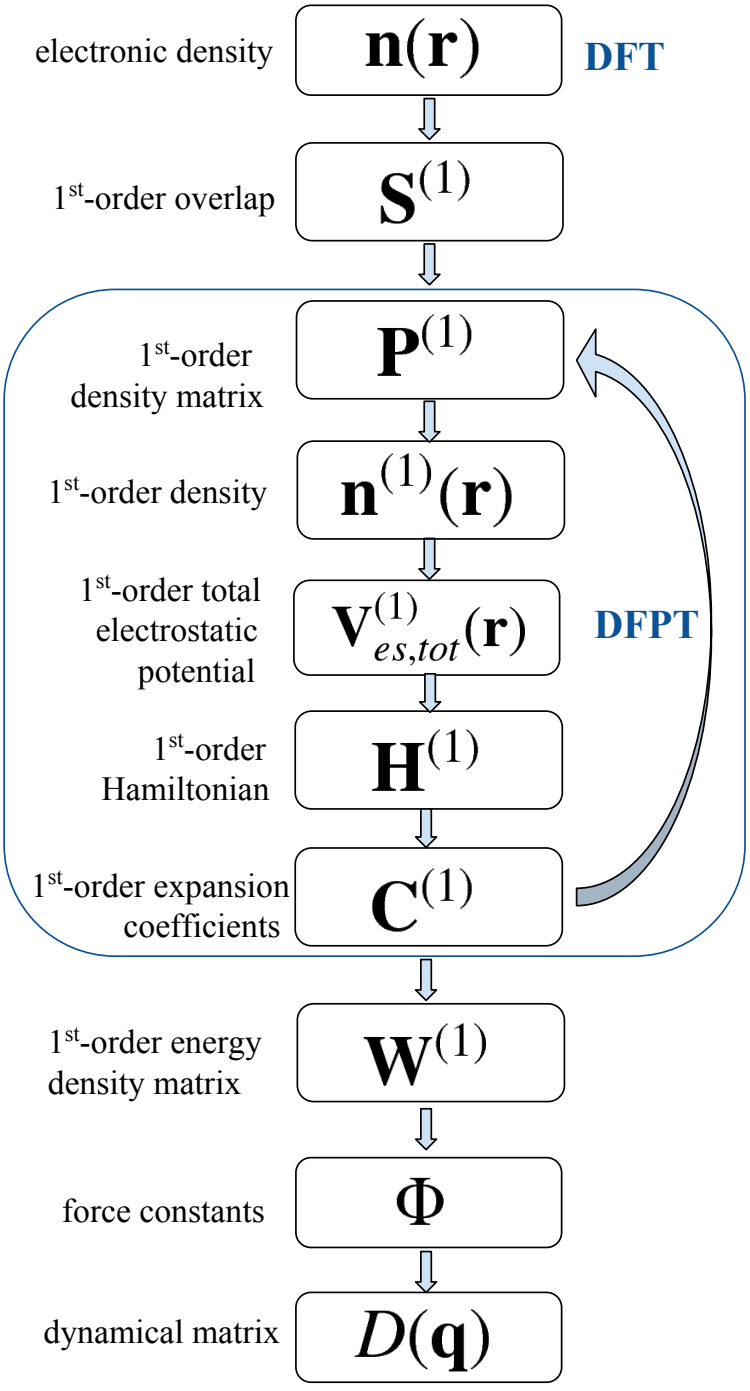


Figure 3.7: Flowchart of the lattice dynamics implementation using a real-space DFPT formalism.

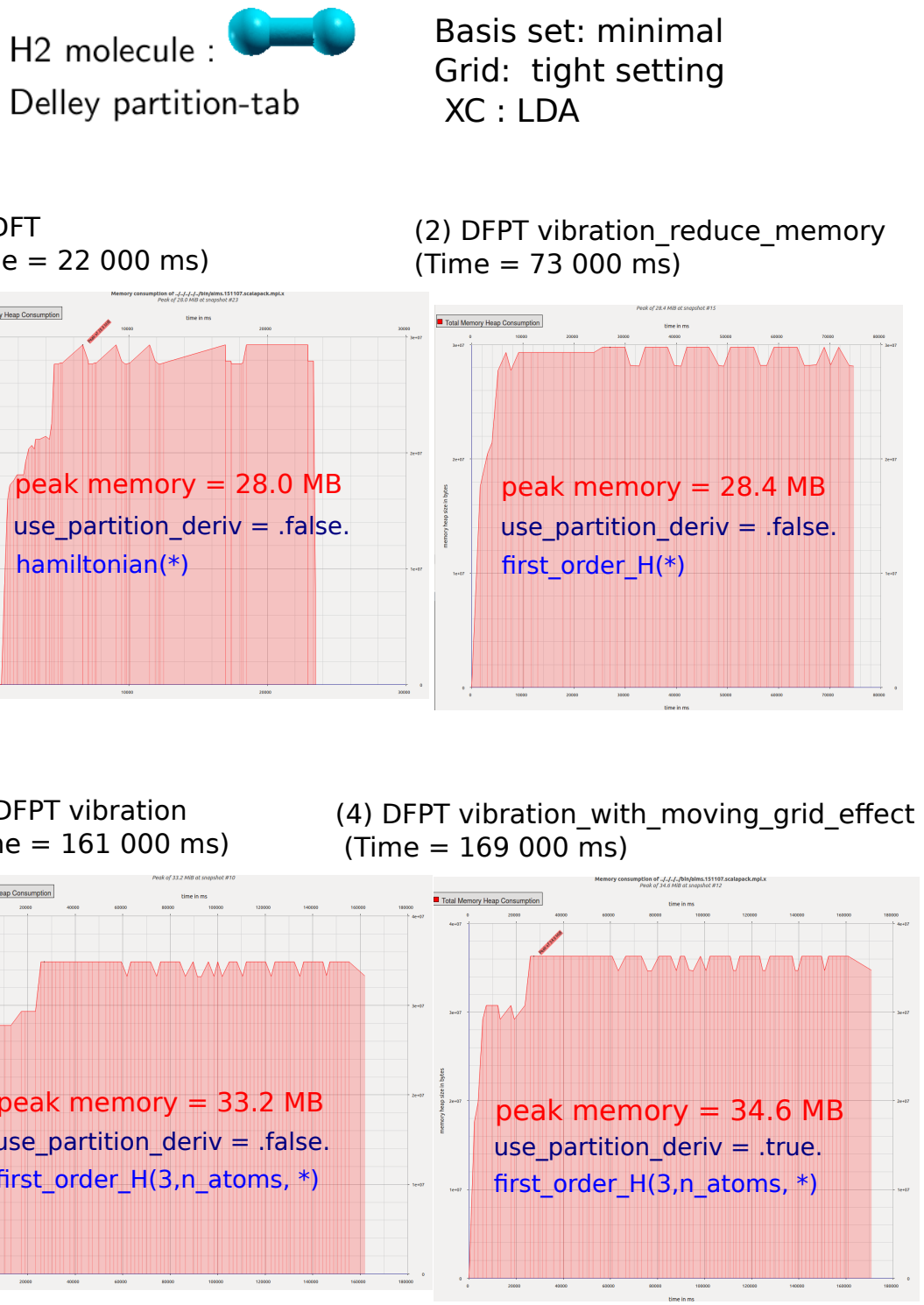


Figure 3.8: The memory profiles using valgrind are shown for DFT and three DFPT vibration keywords. It is clearly shown that the DFPT_vibration_reduce_memory(28.4 MB) code just use nearly the same memory as DFT (28.0MB) calculation, while DFPT_vibration(33.2 MB) and DFPT_vibration_with_moving_gird(34.6 MB) need higher memory because they stored matrix as (3,n_atoms, *).

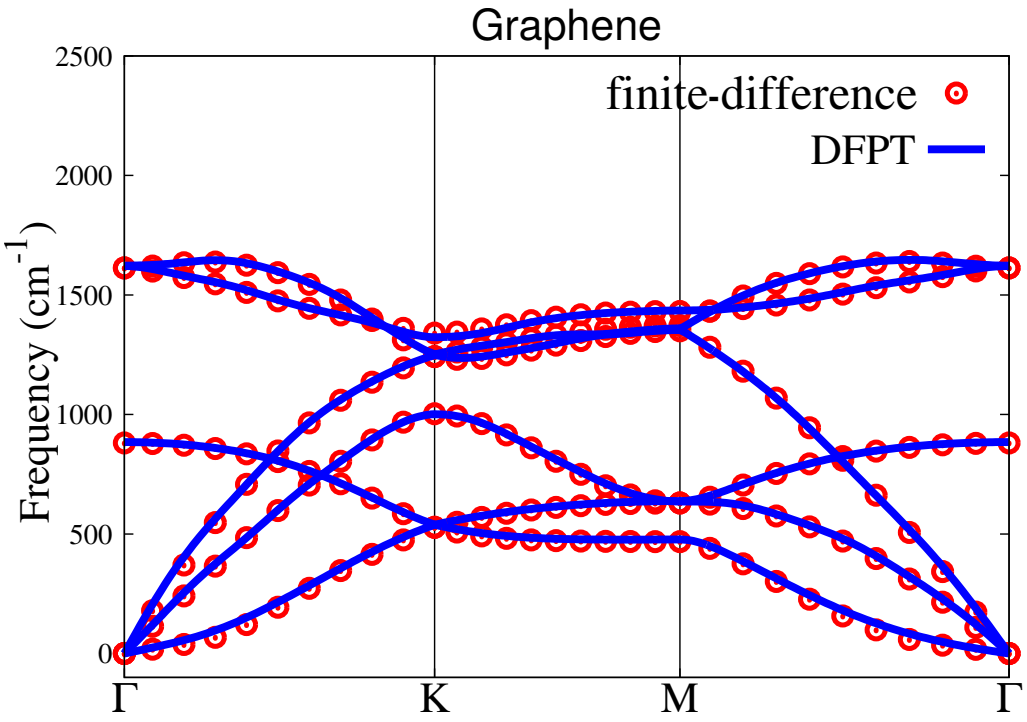


Figure 3.9: Vibrational band structure of graphene computed at the LDA level using both DFPT (solid blue line) and finite differences (red open circles). All calculations have been performed using a 11×11×1 **k**-grid sampling for the primitive Brillouin zone, tight settings for the integration, and a tier 1 basis set.

Homogeneous Electric Fields

Suppose we have an external electrical field ξ, the Hamitonian is changed by adding the following term:

$$H_E = -\mathbf{r} \cdot \xi \tag{3.110}$$

and the induced total energy becomes:

$$E_{tot} = E^0_{tot} - \sum_{I=x,y,z} \mu_I \xi_I - \frac{1}{2} \sum_{I,J} \alpha_{IJ} \xi_I \xi_J \tag{3.111}$$

Here μ_I label the dipole moment,

$$\mu_I = \int n(\mathbf{r}) r_I d\mathbf{r} \tag{3.112}$$

and the corresponding polarizability is defined as the first order derivative of dipole moment with respect to external electrical field :

$$\alpha_{I,J} = \frac{\partial \mu_I}{\partial \xi_J} = \int r_I \frac{\partial n(\mathbf{r})}{\partial \xi_J} d\mathbf{r} \tag{3.113}$$

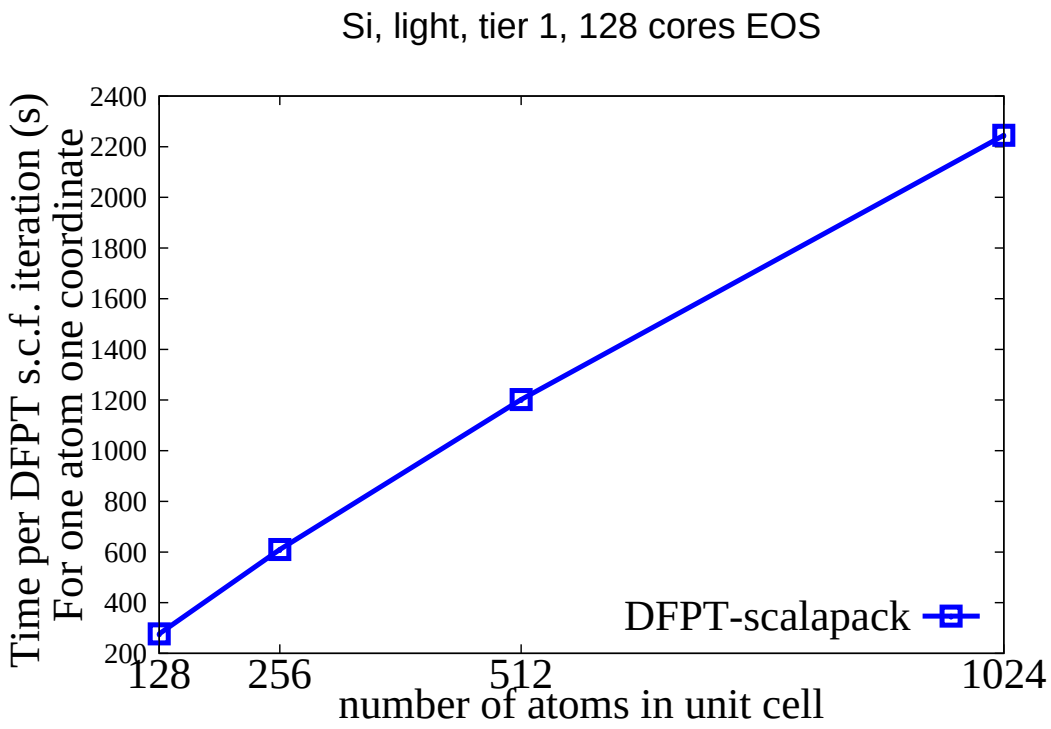


Figure 3.10: The CPUT time per DFPT cycle as a function of the number of atoms in unit cell on 128 CPU cores.

The polarizability for cluster system can be calculated using 3.113 without any problem. However, for extended systems, the position operator is unbounded, in order to deal with it, we use $\langle\psi_i(\mathbf{k})|-\mathbf{r}|\psi_j(\mathbf{k})\rangle = \frac{\langle\psi_i(\mathbf{k})|\bigtriangledown|\psi_j(\mathbf{k})\rangle}{(\varepsilon_i(\mathbf{k})-\varepsilon_j(\mathbf{k}))}$, so we can rewrite the polarizability (Eq.3.113) for extended system as

$$\alpha_{I,J} = \int_{uc} r_I \frac{\partial n(\mathbf{r})}{\partial \xi_J} d\mathbf{r} \tag{3.114}$$

$$= \frac{4}{N_k} \sum_{i,j,\mathbf{k}} \langle\psi_i(\mathbf{k})|\nabla_I|\psi_j(\mathbf{k})\rangle_{uc} \frac{\langle\psi_j(\mathbf{k})|H^{(1)}|\psi_i(\mathbf{k})\rangle_{uc}}{(\varepsilon_j(\mathbf{k})-\varepsilon_i(\mathbf{k}))^2} \tag{3.115}$$

and the corresponding dielectric constant is

$$\epsilon^{\infty}_{IJ} = \delta_{IJ} + \frac{4\pi}{V_{uc}} \alpha_{IJ} \tag{3.116}$$

Similar to the lattice dynamic case, the first order density matrix $\frac{\partial P_{\mu\nu}}{\partial \mathbf{R}_J}$ is needed to be calculated using DFPT cycles. The flowchart of our DFPT cycle for electric field is shown in Fig.3.11.

In order to validate our implementation for extended system, we compare the polarizability with cluster extended method. We use hydrogen line (H_2) as a showcase. All

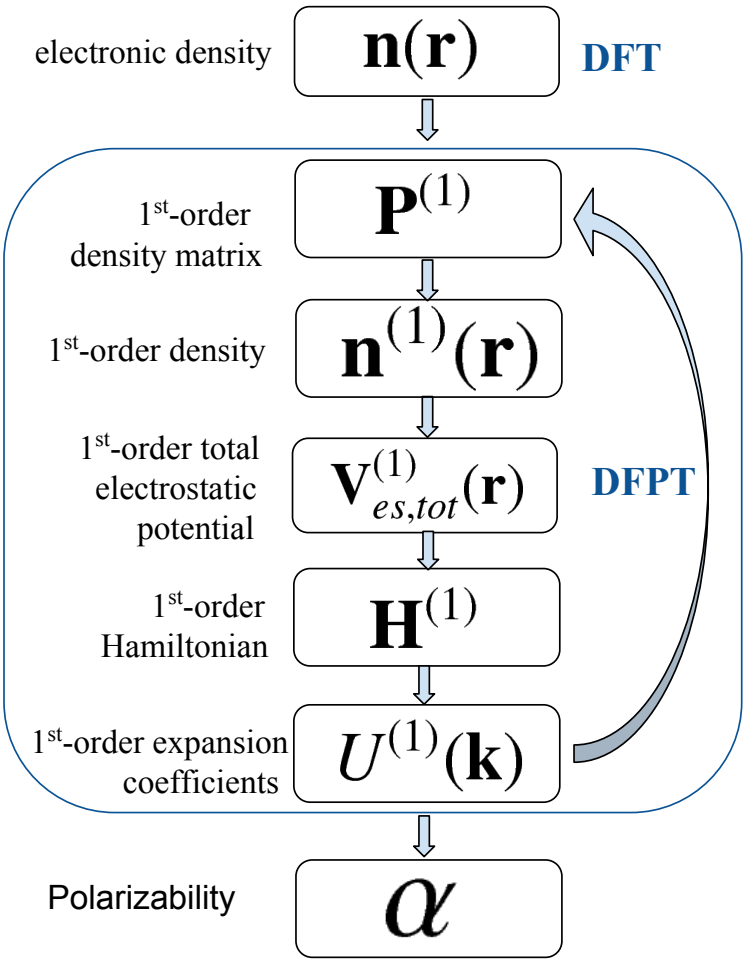


Figure 3.11: Flowchart of the electric field implementation using a real-space DFPT formalism.

calculations have been performed with a geometry shown in Fig.3.12 using the tight integration grids and "minimal" basis sets. For the periodic chain, a reciprocal-space grid of $35 \times 1 \times 1$ electronic **k**-points (in the primitive Brillouin zone) has been utilized as substantiated convergence in Tab. 3.2. The convergence with respect to electronic **k**-points is reasonably fast, in which k grid $35 \times 1 \times 1$ has already get converged with absolute and relative errors of 0.09 Bohr^3 and 0.07 % compared with k grid $70 \times 1 \times 1$.

We fit the DFPT results for $N = 52$ to $N = 64$ with an equation

$$\ln\left(\alpha_N - \alpha_{N-1}\right) = a + \frac{b}{N} \ , \tag{3.117}$$

k	10	20	35	40	70
α_{xx}	180.49	134.76	130.17	130.29	130.26

Table 3.2: The **k**-convergence tes for polarizabllities per H_2 unit cell.

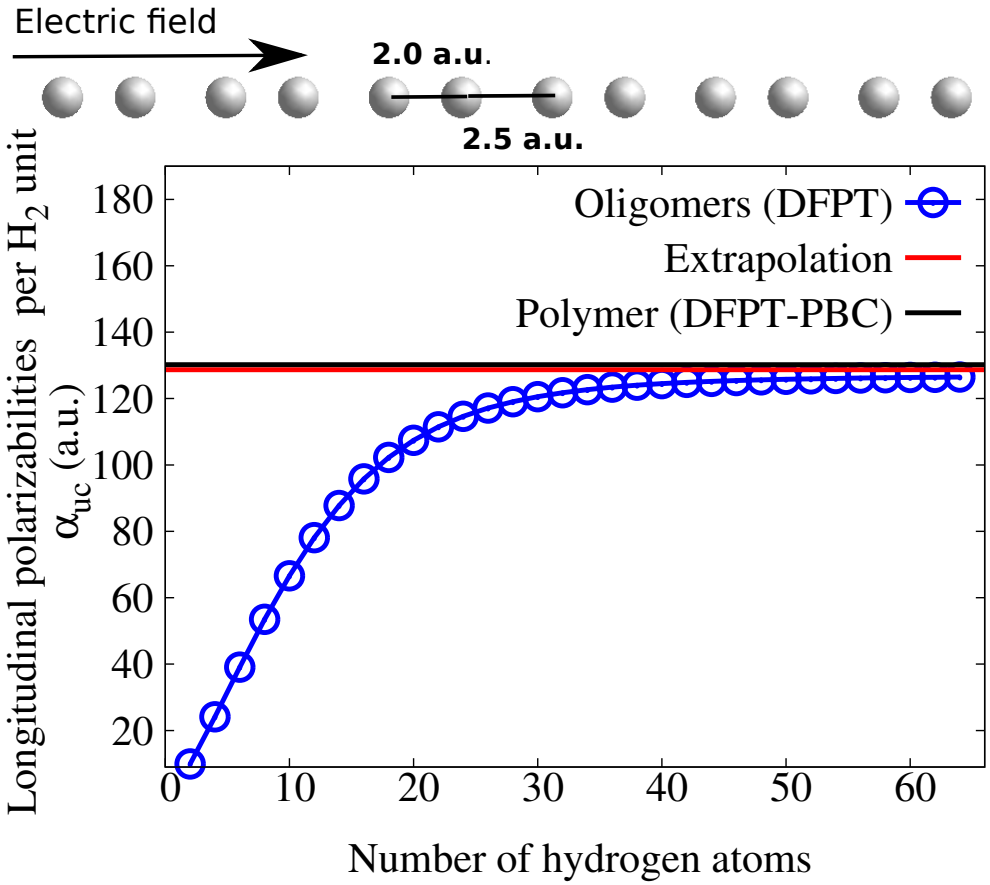


Figure 3.12: Total longitudinal polarizabllities per H_2 unit cell calculated using DFPT with PZ-LDA for molecular hydrogen chains, the bond length is H-H = 2.0 a.u., H_2-H_2 = 4.5 a.u., as shown in the figure. The extrapolated value of α_{uc}^{DFPT} is 128.7, listed with red line. The DFPT-PBC value is 130.2, listed with black line.

so the limiting value of the polarizability per unit cell with N going to infinity is given by:

$$\lim_{N\to\infty} \alpha_{uc} = \exp^{a} \quad . \tag{3.118}$$

We finally get

$$a = 4.857 \tag{3.119}$$
$$b = -1.123 \tag{3.120}$$
$$\tag{3.121}$$

so we have $\alpha(\text{Extrapolation})_{uc} = 128.718$

Some Technical details

Screened Method

In periodic systems, the Coulomb potential is long range, e.g. in a H atom as shown in Fig. 3.13, both $V_{electron-ion}$ and $V^{free}_{electron-electron}$ are (separately) never zero (they decay

H_N (a.u.)	α_N^{DFPT}	α_{uc}^{DFPT}
2	9.923	9.923
4	34.042	24.118
6	73.134	39.092
8	126.619	53.485
10	193.231	66.612
12	271.317	78.086
14	359.107	87.790
16	454.896	95.789
18	557.150	102.254
20	664.553	107.404
22	776.018	111.464
24	890.662	114.644
26	1007.790	117.128
28	1126.853	119.064
30	1247.430	120.577
32	1369.190	121.760
34	1491.883	122.693
36	1615.312	123.429
38	1739.328	124.016
40	1863.812	124.485
42	1988.677	124.865
44	2113.848	125.171
46	2239.272	125.423
48	2364.903	125.631
50	2490.709	125.806
52	2616.657	125.949
54	2742.728	126.071
56	2868.902	126.174
58	2995.166	126.264
60	3121.505	126.339
62	3247.909	126.404
64	3374.371	126.462

Table 3.3: Total longitudinal polarizabllities calculated using PZ-LDA molecular hydrogen chains, with bond length alternation scheme A (H-H = 2.0 a.u., H_2-H_2 = 4.5 a.u. Also we list the longitudinal polarizabllities per H_2 unit cell calculated by $(\alpha_N - \alpha_{N-1})$.

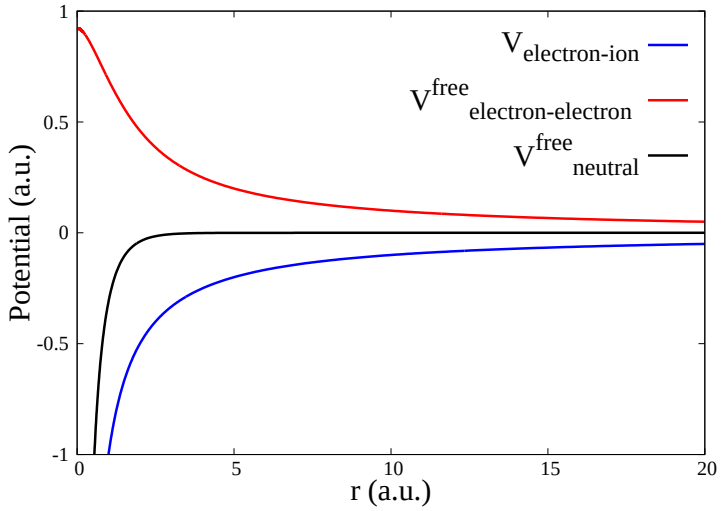


Figure 3.13: The screened scheme can remove long-range tail of Coulomb potential. Here we use H atom as an example.

as $1/r$ but the number of atoms which they interact with in a periodic system grows as r^2 with distance). If we use the neutral free atom potential instead, the potential will approach zero at the radius where the electron charge integrates to exactly -1, which is a finite radius determined by the confinement potential in FHI-aims.

The general idea of screened scheme is to use free part of electronic Coulomb potential to screening the ion charge Coulomb potential and get short-range neutral potential:

$$\begin{aligned} V^{free}_{neutral}(|\mathbf{r}-\mathbf{R}_I|) &= V_{electron-ion} + V^{free}_{electron-electron} \\ &= -\frac{Z_I}{|\mathbf{r}-\mathbf{R}_I|} + \int \frac{n^{free}(\mathbf{r}'-\mathbf{R}_I)}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' \end{aligned} \tag{3.122}$$

So the total electrostatic potential can be written as[28]

$$V_{es}(\mathbf{r}) = -\sum_{Im} \frac{Z_I}{|\mathbf{r}-\mathbf{R}_{Im}|} + \int \frac{n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' \tag{3.123}$$

$$= -\sum_{Im} \frac{Z_I}{|\mathbf{r}-\mathbf{R}_{Im}|} + \int \frac{n^{free}(\mathbf{r}') + \delta n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}'$$

$$= \sum_{I,m} \left[V^{free}_{neutral}(|\mathbf{r}-\mathbf{R}_{Im}|) + \delta V(|\mathbf{r}-\mathbf{R}_{Im}|) \right] \tag{3.124}$$

We follow the line of screened scheme, the first order total electrostatic potential $V^{(1)}_{es,tot}(\mathbf{r})$

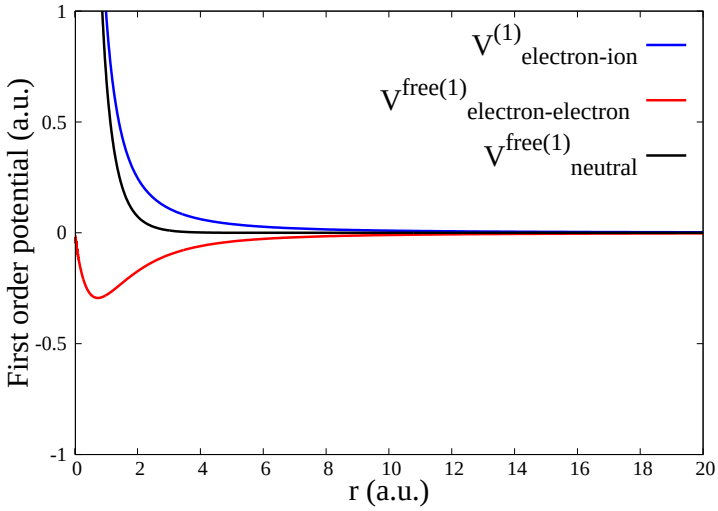


Figure 3.14: The screened method for first order potential. Here we use H atom as an example.

can be written as

$$V^{(1)}_{es,tot}(\mathbf{r}) = -(\sum_{In}\frac{Z_I}{|\mathbf{r}-\mathbf{R}_{In}|})^{(1)} + \int \frac{n^{(1)}(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}' \tag{3.125}$$

$$\begin{aligned} &= \frac{\partial -(\sum_{I,n}\frac{Z_I}{|\mathbf{r}-\mathbf{R}_{In}|})}{\partial \mathbf{R}_{Im}} + \frac{\partial(\int \frac{n^{\text{free}}(\mathbf{r}')+\delta n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}')}{\partial \mathbf{R}_{Im}} \\ &= \frac{\partial V^{\text{free}}(|\mathbf{r}-\mathbf{R}_{Im}|)}{\partial \mathbf{R}_{Im}} + \int \frac{\delta n^{(1)}(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}d\mathbf{r}' \\ &= \frac{\partial V^{\text{free}}(|\mathbf{r}-\mathbf{R}_{Im}|)}{\partial \mathbf{R}_{Im}} + \frac{\partial \delta V(\mathbf{r})}{\partial \mathbf{R}_{Im}} \end{aligned} \tag{3.126}$$

In this way, the first order total electrostatic is divided into two parts: a free part $\frac{\partial V^{\text{free}}(|\mathbf{r}-\mathbf{R}_{Im}|)}{\partial \mathbf{R}_{Im}}$ and residual part $\frac{\partial \delta V(\mathbf{r})}{\partial \mathbf{R}_{Im}}$, as shown in Eq. (3.126).

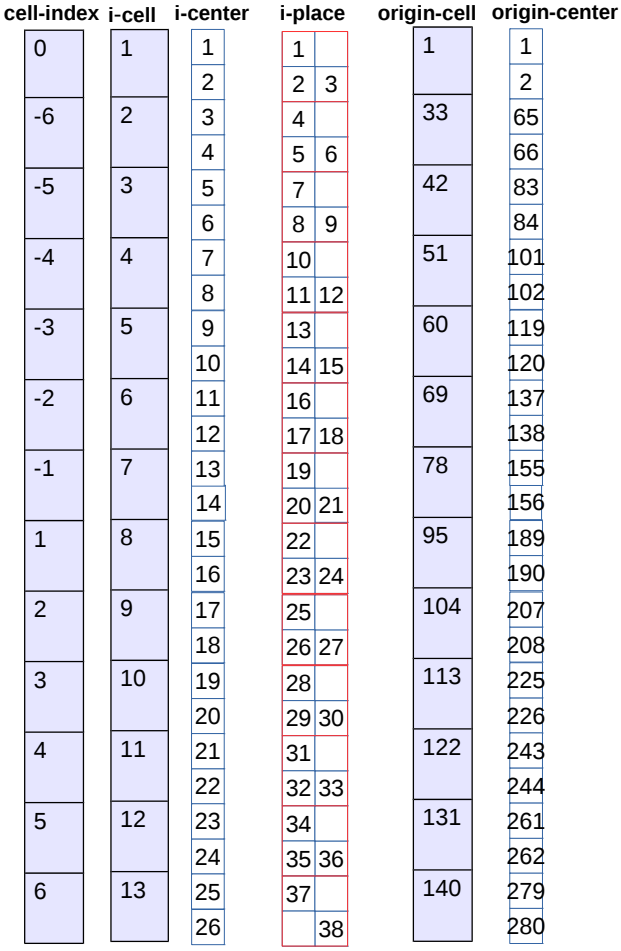
Sparse Matrix

Pleas note that, all matrices in our real-space implementation are in sparse matrix form. see Fig 3.15. We choose the matrix elements which is just in touch with unit cell basis sets, as labelled by i-place in the middle.

Using Pulay mixer

The Pulay mixer is the same as the one used in Magnetic Response. However, in current

Figure 3.15: The sparse matrix storage in FHI-aims for overlap matrix, Hamiltonian matrix and density matrix, here is an example for H_2-line. In total, there are 13 cells (i-cell), with cell-index from [-6,0,0] to [6,0,0]. Only centers (i-center) within these cells are considers to build sparse matrix (i-place), the other original centers are just dropped because of no overlap with unit cell.




```
DFPT phonon_reduce_memory
DFPT polarizability
DFPT dielectric
```

by default, the Pulay mixer (pulay step 8) is used, and the mixing parameter is set by

```
   DFPT_mixing   0.2
   DFPT_sc_accuracy_dm   0.001
```

The Pulay mixer can be changed by writing the number of pulay steps in contron.in.

```
dfpt_pulay_steps 8
```

Tags for general section of `control.in`

Tag: `DFPT vibration` (control.in)

Usage: `DFPT vibration` `[subkeywords and their options]`

Purpose: Allows to calculate vibrations using density-functional perturbation theory, use Acoustic Sum Rule (ASR) to get Hessian matrix, do not use moving-grid-effect.

Usage: `DFPT vibration_with_moving_grid_effect` `[subkeywords and their options]`

Purpose: give the results for vibrations with moving-grid-effect, do not use ASR for Hessian matrix.

Usage: `DFPT vibration_without_moving_grid_effect` `[subkeywords and their options]`

Purpose: give the results for vibrations without moving-grid-effect, ONLY served as comparison with vibration_with_moving_grid_effect.

Tag: `DFPT vibration_reduce_memory` (control.in)

Usage: `DFPT vibration_reduce_memory` `[subkeywords and their options]`

Purpose: Allows to calculate vibrations density-functional perturbation theory by using nearly the same memory as DFT. At present, functionals LDA, PBE are supported, relativistic is also supported. It should be noted that PBE and PBE+TS is supported only for DFPT cycle (first-order-H), but not for Hessian. Only linear-mix (no Pulay-mixer) can be used for DFPT vibration_reduce_memory at present.

Here is an example, the following need to be added to control.in:

```
DFPT vibration_reduce_memory
DFPT_mixing   0.2       #default is 0.2
DFPT_sc_accuracy_dm   1E-6  # default is 1.0d-6
```

Tag: `DFPT phonon_gamma` (control.in)

Usage: `DFPT phonon_gamma [subkeywords and their options]`

Purpose: Allows to calculate phonon for PBC systems using density-functional perturbation theory. This feather use the dense matrix in FHI-aims, which cost a lot of memory, so this keyword only served as a benchmark for DFPT phonon_reduce_memory.

Tag: `DFPT phonon` (control.in)

Usage: `DFPT phonon [subkeywords and their options]`

Purpose: Allows to calculate phonon (real space method) for PBC systems using density-functional perturbation theory. This method could get force constants using real space method and give the phonon band structures. At present, only functionals LDA without relativistic is supported.

Here is an example for using DFPT phonon, the following need to be added to control.in:

```
DFPT phonon
DFPT_mixing   0.5             #default is 0.2
DFPT_sc_accuracy_dm   1.0d-6  # default is 1.0d-3
dfpt_pulay_steps 6            # default is 8
```

Tag: `DFPT phonon_reduce_memory` (control.in)

Usage: `DFPT phonon_reduce_memory [subkeywords and their options]`

Purpose: Allows to calculate phonon (reciprocal space method) at q point for PBC systems using density-functional perturbation theory. At present, this keyword only works to get dynamic matrix at q = 0. This feature is under developing. At present, functionals LDA, PBE are supported, relativistic is also supported. It should be noted that PBE and PBE+TS is supported only for DFPT cycle (first-order-H), but not for Hessian.

Here is an example for using DFPT phonon_reduce_memory, the following need to be added to control.in:

```
DFPT phonon_reduce_memory
DFPT_mixing   0.5             #default is 0.2
DFPT_sc_accuracy_dm   1.0d-6  # default is 1.0d-3
dfpt_pulay_steps 6            # default is 8
```

Tag: `DFPT polarizability` (control.in)

Usage: `DFPT polarizability [subkeywords and their options]`

Purpose: Allows to calculate polarizability for cluster systems using density-functional perturbation theory.
For "DFPT polarizability", functionals LDA, PBE, HF(RI-V) are supported, relativistic is also supported.

Here is an example for using DFPT polarizability, the following need to be added to control.in:

```
DFPT polarizability
DFPT_mixing   0.5             #default is 0.2
DFPT_sc_accuracy_dm   1.0d-6  # default is 1.0d-3
dfpt_pulay_steps 6            # default is 8
```

Tag: `DFPT dielectric` (control.in)

Usage: `DFPT dielectric [subkeywords and their options]`

Purpose: Allows to calculate dielectric constant for extended systems using density-functional perturbation theory. For "DFPT dielectric", functionals LDA, PBE, are supported, relativistic is also supported.

Here is an example for using DFPT dielectric, the following need to be added to control.in:

```
DFPT dielectric
DFPT_mixing   0.5             #default is 0.2
DFPT_sc_accuracy_dm   1.0d-6  # default is 1.0d-3
dfpt_pulay_steps 6            # default is 8
```

Tag: `DFPT_width` (control.in)

Usage: `DFPT_width width`

Purpose: Removes the divergence that can arise in case of small eigenvalue differences and/or fractional occupation numbers. This keyword is to employ in combination with a DFPT dielectric calculation. Note that it has not been thoroughly tested, and some small adjustments might be needed.

`width` is a real number that corresponds to the width of the smearing function (in eV). A value of 0.01 eV proves reasonable in most cases.

The usual expressions employed to calculate the first-order quantities fail when tiny eigenvalue differences are present and/or when the system under study has some fractional occupation numbers, potentially leading to divergences when calculating the polarizability and dielectric constant. In order to circumvent this, we use a similar scheme as the one proposed by de Gironcoli [55], which makes use of smearing functions to convolute the density of states.

Tag: `DFPT_centralised` (control.in)

Usage: `DFPT_centralised boolean`

Purpose: Uses the 2021 unified interface for DFPT. Documentation is in progress and available at https://www.overleaf.com/read/jxtnydzhhxtr . Default is currently False.

3.32 Calculating polarization of solids with FHI-aims

This section describes the relevant keywords connected to the implementation of the Berry-phase formalism within FHI-aims. The theory behind these flags will soon be summarized here.

Tag: `output polarization` (control.in)

Usage: `output polarization pol_direction n_kpoints_dir1 n_kpoints_dir2 n_kpoints_dir3`

Warning: At the current stage, ScalaPACK is not supported, yet. Please specify `KS_method serial` to force the code to use LAPACK instead.

Purpose: Calculates the electronic and ionic contributions to the polarization in periodic systems via the Berry-phase formalism [129] along a specific reciprocal lattice vector pol_direction using a n_kpoints_dir1 $\times$ n_kpoints_dir2 $\times$ n_kpoints_dir3 k-grid. N.B. System must be an insulator/semiconductor.

The polarization direction pol_direction accepts the values 1, 2, or 3, corresponding to the index of the reciprocal space lattice vector along which the polarization is then calculated. The k-point grid defined by the last three fields (n_kpoints_dir1 n_kpoints_dir2 n_kpoints_dir3) should contain integer numbers and defines the k-point density along the reciprocal lattice vectors 1, 2, and 3, respectively. On this mesh, the required eigenstates and wave functions are obtained non-self consistently by Fourier-interpolating the electronic density obtained during the SCF cycle, in close analogy to band-structure or density-of-states calculations.

Note that the polarization is only defined modulo the polarization quantum, therefore, when evaluating polarization differences between different geometries, it is thus necessary to ensure that the polarization changes continuously between different geometries and does not jump by a full quantum. This can be checked by constructing discrete paths between geometries, e.g. between a non-centrosymmetric and centrosymmetric structure. For the latter, the polarization has to vanish. Therefore, relaxed geometries that fulfill all symmetry constraints are necessary to obtain accurate polarization differences.

Multiple polarization calculations can be requested within one FHI-aims run, e.g., to obtain the polarization along all reciprocal lattice vectors:

```
output polarization 1 20 5 5
output polarization 2 5 20 5
output polarization 3 5 5 20
```

For each one of these directives, the polarization is evaluated independently. A summary

for all directives can be found at the end of the polarization calculations, e.g.,

```
Summarizing all directives:
Directive 1 in direction of rec.  latt.  vec.  1 yields the full polarization:
472.792E-03 (C/m^2)
Directive 1 in direction of rec.  latt.  vec.  2 yields the full polarization:
0.0E00 (C/m^2)
Directive 1 in direction of rec.  latt.  vec.  3 yields the full polarization:
0.0E00 (C/m^2)
```

Whenever the polarization is calculated along all three directions within the same run, the code will also output the polarization along the Cartesian axes x, y, z , e.g.,
`Cartesian Polarization 403.809855E-06 -684.500436E-06 10.214219E-03`

This is particularly useful for non-orthogonal lattice vectors, for which (reciprocal) lattice vectors and Cartesian axes do not coincide.

A quick note on convergence: Generally, a much larger density of k-points is required for polarization calculations than for the SCF cycle. In particular along the reciprocal lattice vector pol_direction along which the polarization is evaluated, as the example above shows. The reason is that the Berry-connection is evaluated using finite-differences for the wave-function derivatives with respect to this k-path.

Tag: `output Z2_invariant` (control.in)

Usage: `output Z2_invariant num_of_planes n_kpoints_parallel n_kpoints_perpendicular`

Warning: At the current stage, ScalaPACK is not supported, yet. Please specify `KS_method` serial to force the code to use LAPACK instead.

Purpose: Calculates the evolution of the Wannier Center of Charges (WCC) for num_of_planes planes representing the Brillouin zone. n_kpoints_parallel and n_kpoints_perpendicular denote the k-point mesh in the plane, whereby n_kpoints_parallel are the k-points paralleland n_kpoints_perpendicular are those perpendicular to the direction along which the WCC evolution is evaluated, as in the case of the tag output polarization. The obtained WCC evolution allows the subsequent determination of the topological invariant Z2 [126] and the characterization of insulators into topologically trivial (those with even Z2) and topologically non-trivial (those with odd Z2). Note that the material must be insulating and should have the time reversal symmetry.

The index num_of_planes denotes the following planes in reciprocal space:

1. k_x=[-0.5, 0.5] k_y=[0.0, 0.5] k_z= 0.0 || k_x = kparallel / k_y = kperpendicular
2. k_x= [-0.5, 0.5] k_y=[0.0, 0.5] k_z= 0.5 || k_x = kparallel / k_y = kperpendicular
3. k_x= 0.0 k_y=[-0.5, 0.5] k_z= [0.0, 0.5] || k_y = kparallel / k_z = kperpendicular
4. k_x= 0.5 k_y=[-0.5, 0.5] k_z= [0.0, 0.5] || k_y = kparallel / k_z = kperpendicular
5. k_x= [0.0, 0.5] k_y=0.0 k_z= [-0.5, 0.5] || k_z = kparallel / k_x = kperpendicular
6. k_x= [0.0, 0.5] k_y=0.5 k_z= [0.0, 0.5] || k_z = kparallel / k_x = kperpendicular

The code always iterates over all planes between 1 and num_of_planes and for each plane n the evolution of the WCC can be found in the output file WCCn.dat. For each plane n, the output file WCCn.dat allows to investigate how many times the Berry phase is cycled through the Brillouin zone and thus allows to determine the topological invariant Z2 for this particular plane [126]. In practice, this can be done by plotting the WCC evolution and counting how often the WCC evolution crosses an arbitrary line parallel to the abscissa [243]. An odd number of crossings implies Z2=1 (topological), an even number Z2=0 (trivial insulator). See Fig.3.16 for some examples.

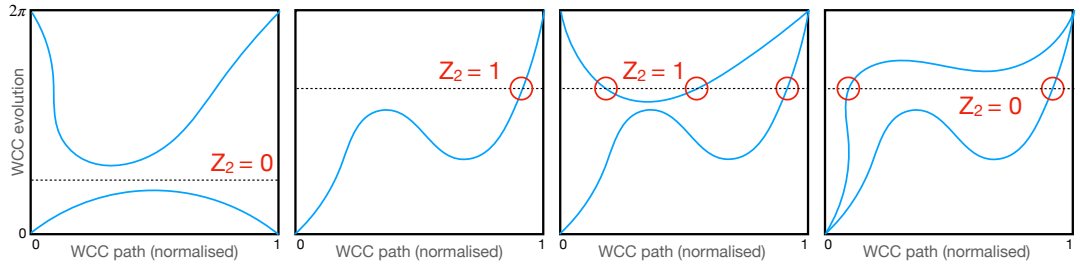


Figure 3.16: Sketches of different WCC evolutions (azure). The number of crossings (red) with an arbitrary horizontal line determines the topological invariant Z2.

Alternatively, the value of Z2 can be determined automatically using the approach proposed by [216] and implemented in the script `get_z2.py` that can be found in the utilities folder of the FHI-aims distribution. The main idea behind the algorithm is that topological indexes (in fact winding numbers) can be determined by tracking the biggest gap in phase between the evolution of the individual WCC centers along the WCC path, see [216, 85]. In practice, it is sufficient to copy `get_z2.py` into the folder containing the WCCn.dat files and to execute it there. Besides plotting the WCC evolution, the determined Z2 indexes for the individual planes n are reported both in the title of the graphes and in the file output_z2.dat.

To distinguish between weak and strong topological insulators [74] Z2 needs to be investigated in multiple planes. For two-dimensional materials with a thick vacuum layer along the Cartesian z axes, it is sufficient to study the WCC evolution in the $k_z = 0$ plane (num_of_planes=1), since there is no dispersion in z direction. For three-dimensional crystals with three-fold rotational symmetry, it is enough to investigate two planes, i.e., one intersecting the Gamma point and one at the border of the Brillouin zone, by using num_of_planes=2. The topological character of the material is then given by the sum of the two obtained indexes modulo 2.

In the general, three-dimensional case, materials are characterized by a set of 4 indexes v_0;v_1,v_2,v_3 and all six planes (num_of_planes=6) in the BZ need to be investigated [74].

The first, strong index v_0 distinguishes between strong and weak topological insulators. The last three indices v_1,v_2,v_3 are called weak topological invariants [73] and denote the value of Z2 for the planes at the BZ boundaries, i.e., those planes in which the index is associated with num_of_planes is 2,4, and 6. For instance, Bi_2Se_3 with $1; 0, 0, 0$ implies that Bi_2Se_3 is a strong topological insulator (there are topologically protected surface states).

How to calculate Born Effective Charges?

Born effective charges (BEC) or dynamic charges are defined as the change in polarization upon the displacement of one atom in the unit cell (and its periodic images, i.e., q=0) as $Z^*_{k,ij} = \frac{\Omega}{e}\frac{\partial P_i}{\partial R_{k,j}}$. Here, the index runs over all atoms in the unit cell, the indexes i,j denote the Cartesian directions for the polarization and the atomic displacement, respectively, Ω is the unit cell volume and e is the elementary charge.
The computation of BECs can be performed using the python script BEC.py that can be found in the FHIaims utilities directory. It utilizes a finite difference approach to determine the derivative of the polarization with respect to atomic displacements.

Example: Displacing Mg atoms along z direction with a k−grid 25×5×2 along direction 1, 2×7×3 direction 2 and 3×4×20 direction 3, finite difference between 0.005 and 0.01Å:

```
python BEC.py -r path-to-aims-excutable --name Mg -p 1 --kx 25 5 2 --ky
2 7 3 --kz 3 4 20 -c 3 -d 0.005 0.01
```

Here, the user has to provide the path to the FHI-aims executable and for which atoms or species the BEC needs to be computed. If the user wants to displace all atoms of the same species, e.g., because they are equivalent, then the option `-p` must be omitted. If only one specific atom from the chosen species should be displaced, then the user has to provide the number of this atom in the geometry.in (`-p 2`). Moreover, the user has to provide the polarization k-grid size along the reciprocal lattice vectors (`--kx --ky --kz`), the Cartesian axis along which the atom should be displaced (`-c` 1,2 or 3 corresponding to x,y, and z), and the displacements in Angstrom to be used for the finite difference (`-d` d1 d2). The latter setting is optional and defaults to `-d` 0 0.0025 if omitted. Values of the displacement of the order of 0.01 Angstrom are usually used in the literature.

The script then performs the following steps in an automatic workflow:

- It reads the provided geometry.in and control.in files and creates two folders, one for each of the chosen displacements d1 and d2 of the chosen atom. In each of these folders it copies the provided control.in and adds the polarization flags with the provided k-grids.
- The polarization is calculated along all Cartesian coordinates with FHI-aims using the specified k-point grids.

- The BECs are obtained via finite difference scheme from the calculated polarizations.

3.33 Molecular Dynamics with Electronic Friction

For questions please directly contact r.maurer@warwick.ac.uk

This module calculates the electronic friction tensor due to the nonadiabatic or electron-phonon coupling. This yields vibrational lifetimes and classical friction forces that can be used in molecular dynamics with nonadiabatic friction in a Langevin formalism, but also as input to calculate electron-phonon coupling constants.

The key references you should consult are:

- Ab-initio tensorial electronic friction for molecules on metal surfaces: nonadiabatic vibrational relaxation R. J. Maurer, M. Askerka, V. S. Batista, J. C. Tully, arXiv:1607.02650 (2016)
- Role of Tensorial Electronic Friction in Energy Transfer at Metal surfaces M. Askerka, R. J. Maurer, V. S. Batista, J. C. Tully, Phys. Rev. Lett. 116, 217601 (2016)

Theory

Typically, for molecular dynamics at ambient conditions, the Born-Oppenheimer approximation is well justified. This means that the time scales of electronic and nuclear motion are well separated. This is the case for most thermal reactions of molecules in gas phase or for insulating or semi-conducting materials. Therefore nuclei can be viewed as moving on a single potential energy surface (PES) given by the ground state electronic structure. This is, however, not the case for nuclei moving on or in metal surfaces (see Fig. 3.17). The continuum of electronic states in metals can already be excited by the vibrational motion of the adsorbate atoms due to resonance at similar energy scales. As a result adsorbate atoms exhibit additional frictional forces. Another way to view this is that the vibrations or phonons are being screened by electronic excitations giving them a finite lifetime. Assuming that the ground and excited state PES are parallel and that these adiabatic effects are only weakly perturbing the adsorbate nuclear motion, we can apply perturbation theory to this problem.

The picture in Fig. 3.17 holds if:

- the coupling is weak compared to the individual contributions of electrons and nuclei, and if
- electron-hole pair excitations do not lead to a qualitative change in the nuclear dynamics.

Following first order time-dependent perturbation theory or many-body perturbation theory (using lowest order RPA) one arrives at the Fermi's golden rule expression for relaxation rate along a vibration with frequency ω_j:

$$\Gamma(\omega_j) = \frac{\pi\hbar^2\omega_j}{M} \sum_{\mathbf{k},\nu,\nu'>\nu} |g^j_{\mathbf{k}\nu,\nu'}|^2 \cdot [f(\epsilon_{\mathbf{k}\nu}) - f(\epsilon_{\mathbf{k}\nu'})] \cdot \delta(\epsilon_{\mathbf{k}\nu'} - \epsilon_{\mathbf{k}\nu} - \hbar\omega_j), \qquad (3.127)$$

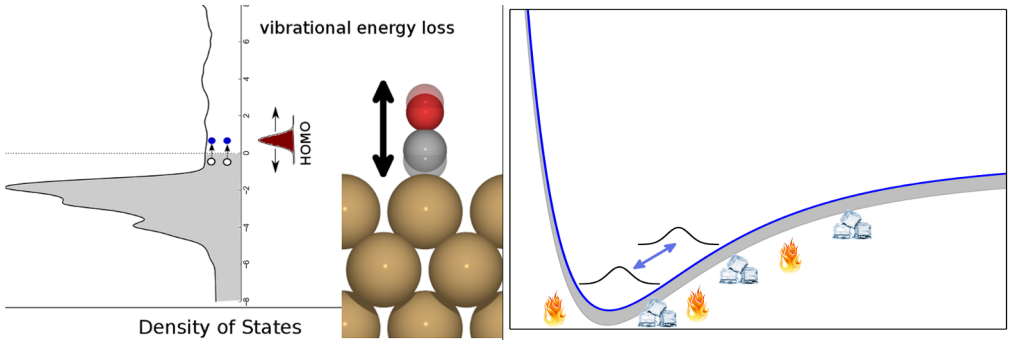


Figure 3.17: left: Schematic view of adsorbate vibration (here shown for a CO molecule on a Cu(100) surface) leading to changes in the electronic structure that excite electron-hole pairs from below to above the Fermi level of the metal density-of-states. right: In the Langevin picture of electronic friction, these electron-hole pairs act as energy gain or loss channels on the nuclear motion along a single potential energy surface.

with

$$g^{j}_{\mathbf{k}\nu,\nu'} = \langle \psi_{\mathbf{k}\nu} | \mathbf{e}_j \cdot \nabla_{\mathbf{R}} | \psi_{\mathbf{k}\nu'} \rangle \,. \tag{3.128}$$

The relaxation rate Γ defines the lifetime of the mode $\tau = 1/\Gamma$ and the vibrational linewidth $\gamma = \Gamma\hbar$. In eq. 3.128, $\mathbf{e}_j$ is the atomic displacement vector corresponding to the vibrational normal mode ω_j and $\nabla_{\mathbf{R}}$ is the vector of Cartesian derivatives.

This can be rewritten in terms of cartesian displacements.

$$\Gamma(\omega_j) = \frac{1}{M}\mathbf{e}_j^T \cdot \left(\pi\hbar \sum_{\mathbf{k},\nu,\nu'>\nu} \langle \psi_{\mathbf{k}\nu} | \frac{\partial}{\partial R_{n'a'}} | \psi_{\mathbf{k}\nu'} \rangle \langle \psi_{\mathbf{k}\nu'} | \frac{\partial}{\partial R_{na}} | \psi_{\mathbf{k}\nu} \rangle \cdot (\epsilon_{\mathbf{k}\nu'} - \epsilon_{\mathbf{k}\nu}) \cdot \right. \tag{3.129}$$

$$\left. [f(\epsilon_{\mathbf{k}\nu}) - f(\epsilon_{\mathbf{k}\nu'})] \cdot \delta(\epsilon_{\mathbf{k}\nu'} - \epsilon_{\mathbf{k}\nu} - \hbar\omega_j) \right) \cdot \mathbf{e}_j$$

Correspondingly the rate of decay of adsorbate motion along a displacement vector $\mathbf{e_j}$ is given by:

$$\Gamma(\omega_j) = \mathbf{e}_j^T \cdot \tilde{\mathbf{\Lambda}}(\omega_j) \cdot \mathbf{e}_j. \tag{3.130}$$

where $\tilde{\Lambda}$ is the so-called mass-weighted friction tensor or tensor of nonadiabatic relaxation rates $\tilde{\Lambda}_{ij} = \Lambda_{ij}/(\sqrt{m_i}\sqrt{m_j})$.

In practice the friction tensor is calculated using the ground state Kohn-Sham eigenstates and evaluated in the quasi-static or zero-frequency limit as $\Lambda(\omega \to 0)$. This can be done for a cluster and a periodic system. In the periodic case, the relaxation rate of periodic motion ($q = 0$, default) can be calculated or, in combination with DFPT, averaged over the whole phonon Brillouin zone ($\int_{\mathbf{q}} \mathbf{\Lambda}(\mathbf{q})\delta(\omega - \omega_{\mathbf{q}})$ (work in progress). The matrix elements are expressed in the local atomic orbital basis by derivatives with respect to the Hamiltonian and overlap matrix in basis representation (see references for more details).

The implementation in FHI-Aims collects all excitations from occupied to unoccupied states in a window of `friction_max_energy` close to the Fermi level into a discretized electron-phonon spectral function (with discretization length `friction_discretization_length`

) and applies a Gaussian broadening of friction_broadening_width to facilitate convergence with respect to $\mathbf{k}$-points. The corresponding spectral functions along every pair of cartesian components can be printed in file 'friction_gamma2.out', but as default only the final friction tensor evaluated at zero frequency is printed in 'friction_tensor.out'. Calculations need to be converged with respect to the number of k points. Convergence is achieved if the relaxation rate is stable over a large range of friction_broadening_width values (10% variation over 0.3-0.6 eV).

In order to do this, first the coupling matrix elements need to be calculated. This can be done using finite difference displacements of the involved atoms or Density Functional Perturbation Theory (see calculate_friction).

Calculation workflow and functionality

The electron-phonon calculation in FHI-aims involves following steps:

1. SCF calculation.
2. Finite-difference or DFPT calculation of matrix elements. Optionally, these matrix elements can be read from files.
3. Calculate relaxation rates and line widths from electronic structure data.

The module currently allows to:

- calculate the nonadiabatic relaxation rate tensor in the quasi-static limit using finite-difference matrix elements for cluster and PBC systems. The latter only works for the Gamma point ($q = 0$).
- atomwise input and output nonadiabatic coupling matrix elements, which can serve as a restart mechanism
- DFPT evaluation of overlap derivatives for the cluster case. Coupling to the DFPT module an extension to phonon wavevectors beyond the $q = 0$ point is work in progress.

Tags for general section of `control.in`

Tag: `calculate_friction` (control.in)

Usage: `calculate_friction type`
Purpose: Triggers the calculation of the friction tensor.
`type`: String that defines the type of calculation to be performed.

- numerical_friction: Using finite difference to calculate matrix elements. default WARNING: Some errorneously large friction tensor elements may be present if a friction atom lies on a symmetry plane.
- DFPT: Using Density Functional Perturbation Theory to calculate matrix elements (Currently not well tested, not recommended for metallic systems or fractional occupations)

With the keyword `type` the user can specify the calculation mode for the evaluation of electron-phonon coupling matrix elements.
In addition, the keyword `empty_states` should be set to a large number, or the keyword `calculate_all_eigenstates` should be used, to make sure the code generates all possible empty states provided from the basis set. This number will also be reduced automatically by the code to the maximum number that can be generated from the basis set.

ScaLAPACK support is only valid for periodic systems currently.

Tag: `friction_numeric_disp` (control.in)

Usage: `friction_numeric_disp disp`
Purpose: This keyword provides the finite difference displacement stencil `disp` in Å for numerical calculation of the friction tensor. The default is 0.0025 Å.

Tag: `friction_broadening_width` (control.in)

Usage: `friction_broadening_width width`
Purpose: This keyword specifies the width of the broadening function that is used to average over the spectral function and to facilitate Brillouin zone integration. Affects friction tensor only (doesn't affect functions outputted as a response in energy). The typical range is 0.3 to 0.6 eV. The default value is 0.3 eV.

Tag: `friction_temperature` (control.in)

Usage: `friction_temperature float`
Purpose: This keyword specifies the electronic temperature in Kelvin that enters through state occupations in Fermi's rolden rule. The default value is 300 K.

Tag: `friction_iter_limit` (control.in)

Usage: `friction_iter_limit integer`
Purpose: This keyword specifies the maximum number of iterations in the finite-difference or DFPT evaluation of electron-phonon matrix elements. The default value is 20.

Tag: `friction_accuracy_rho` (control.in)

Usage: `friction_accuracy_rho float`
Purpose: This keyword specifies the accuracy to which the electronic density is converged in the finite difference calculation. The default is 1d-5 $e/a_0{}^3$.

Tag: `friction_accuracy_etot` (control.in)

Usage: `friction_accuracy_etot float`
Purpose: This keyword specifies the accuracy to which the electronic energy is converged in the finite difference calculation. The default is 1d-5 eV.

Tag: `friction_accuracy_eev` (control.in)

Usage: `friction_accuracy_eev float`
Purpose: This keyword specifies the accuracy to which the sum of Kohn-Sham eigen energies is converged in the finite difference calculation. The default is 0.01 eV.

Tag: `friction_accuracy_potjump` (control.in)

Usage: `friction_accuracy_potjump float`
Purpose: This keyword specifies the accuracy to which the potential is converged in the finite difference calculation. The default is 1.01 eV.

Tag: `friction_delta_type` (control.in)

Usage: `friction_delta_type type`
Purpose: This keyword specifies the shape of the broadening function that is used to generate a converged electron-phonon spectral function. The default type is a gaussan function. Choices include a gaussian function (`gaussian`, a square window (`square`), a squashed Fermi function (`squashed_fermi`), a Lorentzian (`lorentzian`), and a sine function (`sine`).

Tag: `friction_double_delta` (control.in)

Usage: `friction_double_delta boolean`
Purpose: If set to true, evaluates the friction tensor in the low temperature limit expression given in Ref 1 of this section. Here the Fermi occupation factor is not included. Two delta functons are employed with widths controlled by `friction_broadening_width` . This feature is experimental.
`boolean`: .true. or .false.

Tag: `friction_max_energy` (control.in)

Usage: `friction_max_energy energy`
Purpose: This keyword sets the maximum excitation energy in eV for which electronic excitations will be included. The safe default is 4 times the value of `friction_broadening_width` . It should not be below 1 eV for a reliable broadened friction tensor.

Tag: `friction_coupling_matrix_mode` (control.in)

Usage: `friction_coupling_matrix_mode integer`
Purpose: This keyword sets the expression with which the coupling matrix elements are constructed from first order overlap and first order hamiltonian. There are three different modes (0 = "Head-Gordon-Tully-type", 1="Averaged-type", 2="exact matrix elements", 3="Effective potential type"). The default is 0. Type 2 only works in combination with DFPT. Type 3 is experimental. For details on the different types, please refer to the above publication reference. There is significantly less computational expense when using Type 0 for large systems.

Output of spectral function

Various levels of the electron phonon response output are provided for convenience. `friction_output_couplings` provides the full tensorial non-adiabatic coupling elements, $|g^j_{\mathbf{k}\nu,\nu'}|^2$ along with their raw excitation energies. `friction_output_gamma` provides the nonadiabatic coupling element, $g^j_{\mathbf{k}\nu,\nu'}$ response as a function of energy, discretised on a grid using a (small) finite gaussian smearing. `friction_output_gamma2` provides the tensorial response as a function of energy, $\tilde{\Lambda}_{ij}(\epsilon)$, similary smeared using gaussian function.

Tag: `friction_output_couplings` (control.in)

Usage: `friction_output_couplings boolean`
Purpose: This keyword controls the output of raw nonadiabatic coupling elements / ps^{-1} as a function of excitation energy / eV. For periodic systems this generates 1 file per k point, for clusters there is 1 file per unique atomic coordinate. Currently not valid for ScaLAPACK calculations.
`boolean`: .true. or .false.

Tag: `friction_output_gamma2` (control.in)

Usage: `friction_output_gamma2 boolean`
Purpose: Currently this keyword controls the output of the electron-phonon response / ps^{-1} as a function of energy / eV. Excitations are collected and smeared using gaussian broadening, width controlled by `friction_window_size` . Currently not valid for ScaLAPACK calculations. Subject to future change
`boolean`: .true. or .false.

Tag: `friction_output_gamma` (control.in)

Usage: `friction_output_gamma boolean`
Purpose: Currently this keyword controls the output of the nonadiabatic coupling response / $Å^{-1}$ for each atomic coordinate as a function of energy / eV. Excitations are collected and smeared using gaussian broadening, width controlled by `friction_window_size` . Currently not valid for ScaLAPACK calculations. Subject to future change
`boolean`: .true. or .false.

Tag: `friction_window_size` (control.in)

Usage: `friction_window_size width`
Purpose: This keyword sets the broadening width when calculating the full vibronic spectral function with `friction_output_gamma2` or `friction_output_gamma` . There is typically no need to change this value. The default value is 0.01 eV.

Tag: `friction_discretization_length` (control.in)

Usage: `friction_discretization_length width`
Purpose: This keyword sets the discretization of the grid on which the electron-phonon response will be printed with keyword `friction_output_gamma2` and `friction_output_gamma` . The default value is 0.01 eV.

Tag: `friction_output_jdos` (control.in)

Usage: `friction_output_jdos boolean`
Purpose: Currently this keyword if set to true will output the joint-density of states / eV^{-1} for first order interband transitions (i.e those within the same k point). The JDOS is k weighted. It reflects the density of transitions used within the friction calculation.
`boolean`: .true. or .false.

Input and Output of matrix elements and vectors

Tag: `friction_read_matrices` (control.in)

Usage: `friction_read_matrices boolean`
Purpose: This keyword leads to skipping the evaluation of electron-phonon matrix elements. FHI-aims will read the matrix elements from files, generated by the keyword `output friction_matrices` . Currently not valid for ScaLAPACK calculations.
`boolean`: .true. or .false.

Tag: `output friction_matrices` (control.in)

Usage: `output friction_matrices`
Purpose: Prints first order hamiltonian / Ha a_0^{-1} and first order overlap matrices / a_0^{-1} to file. This can be read via `friction_read_matrices` . Currently not valid for ScaLAPACK calculations.

Tag: `output friction_eigenvectors` (control.in)

Usage: `output friction_eigenvectors`
Purpose: Prints friction eigenvectors as jmol readable file.
`boolean`: .true. or .false.

Tags for `geometry.in`

Tag: `calculate_friction` (geometry.in)

Usage: `calculate_friction boolean`

Purpose: In `geometry.in`, includes the last specified `atom` into the calculation of the friction tensor.

`boolean` is .true. or .false., Default: `.false.`

3.34 Linear macroscopic dielectric function and Kubo-Greenwood transport

The linear macroscopic dielectric tensor $\epsilon_{ij}(\omega)$, the ratio between the average of the total potential in one unit cell and the external field, within the RPA is calculated. For derivation from the microscopic dielectric function $\epsilon(\vec{r},t;\vec{r'},t')$ and references see [3] (Chapter 1, 2 and appendix). From the complex frequency dependent dieletric function all other optical constants can be determined, e.g. optical conductivity, Loss function, Reflectivity, etc. Also see Appendix K in Ashcroft/Mermin: solid state physics.
The frequency dependent real and imaginary part of the inter- (eq. 3.132) and intra- (eq. 3.131) band contribution to the linear dielectric tensor is calculated as post-processing after convergence of the SCF-cycle from (atomic units):

$$\epsilon_{ij}(\omega) = \delta_{i,j} - \frac{4\pi}{V_{cell}\omega^2}\sum_{n,\mathbf{k}}\left(-\frac{\partial f_0}{\partial \epsilon}\right)_{\epsilon_{n,\mathbf{k}}} p_{i;n,n,\mathbf{k}}p_{j;n,n,\mathbf{k}} \tag{3.131}$$

$$+\frac{4\pi}{V_{cell}}\sum_{\mathbf{k}}\sum_{c,v}\frac{p_{i;c,v,\mathbf{k}}p_{j;c,v,\mathbf{k}}}{\left(\epsilon_{c,\mathbf{k}}-\epsilon_{v,\mathbf{k}}-\omega\right)\left(\epsilon_{c,\mathbf{k}}-\epsilon_{v,\mathbf{k}}\right)^2} \tag{3.132}$$

f is the Fermi-function and V_{cell} the unit cell volume. The intra band part is singular at $\omega = 0$. Here the plasma frequency $\omega_{pl;ij}$ is calculated:

$$\omega^2_{pl;ij} = \frac{1}{\pi}\sum_n \int_{\vec{k}} p_{i;n,n,\vec{k}}p_{j;n,n,\vec{k}}\delta(\epsilon_{n,\vec{k}}-\epsilon_F) \tag{3.133}$$

By adopting a Drude-like shape for the intra-band contributions a lifetime broadening Γ is introduced and $\epsilon^{inter}_{ij}(\omega)$ becomes:

$$\mathrm{Im}(\epsilon^{intra}_{ij}(\omega)) = \frac{\Gamma\omega^2_{pl;ij}}{\omega(\omega^2+\Gamma^2)} \tag{3.134}$$

$$\mathrm{Re}(\epsilon^{intra}_{ij}(\omega)) = 1 - \frac{\omega^2_{pl;ij}}{\omega^2+\Gamma^2} \tag{3.135}$$

In the case of a spin unpolarized calculation eq. 3.132 and eq. 3.131 have to be multiplied by 2 to yield the correct occupation.
The following quantities are needed/calculated:

- $\epsilon_{n,\vec{k}}$ - the eigenvalue of the Kohn-Sham eigenstate $(n,\vec{k})$.
- $\delta(\epsilon_{n',\vec{k}}-\epsilon_{n,\vec{k}}-\omega) = \frac{1}{\sqrt{2\pi}width}exp\left(-\frac{1}{2}\frac{(\epsilon_{n',\vec{k}}-\epsilon_{n,\vec{k}}-\omega)^2}{\Gamma^2}\right)$ - Gauss function with width $\Gamma{=}width$ for calculating the plasma frequency. In eq 3.132 ω is replaced by $\omega+\mathrm{i}\Gamma$, introducing Lorentzian broadenig.
- $p_{j;n',n,\vec{k}} = \left\langle\psi_{n'\vec{k}}|\nabla_j|\psi_{n\vec{k}}\right\rangle$ - the momentum matrix elements calculated form the real space basis functions in k-space and KS-eigenstate basis

$$\left\langle\psi_{n'\vec{k}}|\nabla_j|\psi_{n\vec{k}}\right\rangle = \sum_{ij} c^{*\vec{k}}_{in'}c^{\vec{k}}_{jn}\sum_{\vec{N},\vec{M}} \mathrm{e}^{\mathrm{i}\vec{k}\left[\vec{T}(\vec{N})-\vec{T}(\vec{M})\right]}\left\langle\phi_{i\vec{M}}|\nabla_j|\phi_{j\vec{N}}\right\rangle \tag{3.136}$$

$$\langle \phi_{i\vec{M}} | \nabla | \phi_{i\vec{N}} \rangle = \int_{\text{unit cell}} \mathrm{d}^3 r \phi_{i,\vec{M}} \vec{\nabla} \phi_{j,\vec{N}} \tag{3.137}$$

with the real space functions $\phi_i(\vec{r})$ centered in unit cells shifted by $\vec{T}\left(\vec{N}\right)$, $\vec{N} = (N_1, N_2, N_3)$, see [28] for details.

Building up on the expressions for the dielectric function also the corresponding Kubo-Greenwood transport properties expressed via the Onsager coefficients L_{ij}:

$$L_{ij}(\omega) = \frac{2\pi e^{4-i-j}}{3Vm_e^2\omega} \sum_{\boldsymbol{k},m,n} |\langle \Psi_m | \hat{\boldsymbol{p}} | \Psi_n \rangle|^2 \cdot \left(\frac{\epsilon_{\boldsymbol{k}m} + \epsilon_{\boldsymbol{k}n}}{2} - \epsilon_{Fermi} \right)^{i+j-2} \cdot (f_{\boldsymbol{k}m} - f_{\boldsymbol{k}n}) \, \delta \left(\epsilon_{\boldsymbol{k}n} - \epsilon_{\boldsymbol{k}m} - \hbar\omega \right) \tag{3.138}$$

In this representation L_{11} coresonds to the optical conductivity $\sigma(\omega)$. Furthermore the (frequency dependent) Seebeck coefficient is easily obtainable:

$$S = \frac{L_{12}}{TL_{11}} \tag{3.139}$$

Tags for general section of `control.in`:

Tag: `compute_dielectric` (control.in)

Usage: `compute_dielectric` ω_{max} n_{ω}

Purpose: Sets basic parameters for calculating the imaginary and real part of the inter-band and intra-band contribution to the linear dielectric tensor within the RPA approximation. This keyword is specified once to set parameters for the dielectric tensor calculation.

By setting this keyword, the whole dielectric tensor components (in directions: x_x, y_y, z_z, x_z, y_z, x_y) would be output automatically within the corresponding absorption coefficients (in directions: x_x, y_y, z_z) for the diagonal parts.
The momentum matrix elements in the energy window [VBM-(ω_{max} + 10.0 eV), CBM+(ω_{max} + 10.0 eV)] (in eV) relative to the internal zero will be summed.
The default broadening type and broadening width used for the delta function is 0.1 eV in Lorentzian type. These settings can also be changed via the `dielectric_broadening` keyword.
In order to avoid numerical integration errors caused by including 0 eV energy, the minimum energy (ω_{min}) are automatically setting to a spefic value corresponding to the broadening width you used (broadening width/ 10.0).
The resulting output files will be named dielectric_function_(directions).out (e.g. dielectric_function_x_x.out for the x_x direction) and absorption_(directions).out.
In order to test the impacts of different broadening type and broadening parameters, the individual tensor component for the dielectric constants can also be specified via the `output dielectric` keyword. By setting this keyword, the code would only output the dielectric functions in the directions you listed in the `output dielectric` keyword, instead of automatically outputing the whole tensor components. The calculation is very sensitive to the k-point grid, an extremely high number of k-points might be needed for convergence, especially for metals.

Tag: `dielectric_broadening` (control.in)

Usage: `dielectric_broadening` `broadening_method` `width`

Purpose: Changing the broadening function type and broadening parameters used in the dielectric calculation. To use this keyword, the `compute_dielectric` keyword must be specified in `control.in`
The Delta-distribution is approximated by a broadening function specified by the `broadening_method` option with a defined width specified by the `width` option (in eV). Gaussian (`gaussian`) and lorentz (`lorentzian`) broadenings are supported.

Tag: `output dielectric` (control.in)

Usage: `output dielectric broadening_method width i j`

Purpose: Output the `ij` tensor components (choices: i,j = x, y, z) of the imaginary and real parts of the inter-band and intra-band contribution to the linear dielectric tensor. This keyword may be specified multiple times in `control.in` to output more than one tensor component (possibly with different broadenings) per calculation. The RPA approximation (i.e. Lindhard theory) is used. To use this keyword, the `compute_dielectric` keyword must be specified in `control.in`.
Note: This keyword are just setted for testing purpose. By setting this keyword, only the directions you listed will be output. The Delta-distribution is approximated by a broadening function specified by the `broadening_method` option with a defined width specified by the `width` option (in eV). Gaussian (`gaussian`) and Lorentz (`lorentzian`) broadenings are supported. Good starting choices for parameters are `broadening_method` = gaussian and `width`=$0.05eV$. We encourage the user to try out different broadenings by specifying this keyword multiple times.

Tag: `compute_absorption` (control.in)

Usage: `compute_absorption width` E_{min} E_{max} ω_{min} ω_{max} n_ω `i use_gauss`

Purpose: Calculate and output the `i` component (choices: x, y or z) of the linear absorption.

$$\alpha_i(\omega) = \frac{8\pi^2}{\omega V_{cell}} \sum_{c,v} \sum_{\vec{k}} \left| p_{i;c,v,\vec{k}} \right|^2 \delta(\epsilon_{c,\vec{k}} - \epsilon_{v,\vec{k}} - \omega) \mathrm{d}\vec{k} \qquad (3.140)$$

The momentum matrix elements in the energy window [E_{min},E_{max}] (in eV) relative to the internal zero are summed up. The Delta-distribution is represented by a Gaussian function (`use_gauss` = .true.) or a Lorentz function (`use_gauss` = .false.) with width `width` (in eV) for n_ω ω-values in the interval [ω_{min}, ω_{max}] (in eV). The output file is named *absorption_*`i`*.out*. A good choice is: `width`=$0.1eV$, usually it is enough to include only a few states below and above the fermi level in the energy window [E_{min},E_{max}]. The unit is a_0^{-1}. V_{cell} the unit cell volume. (see page 38, eq. 3.13 and 3.14 of http://www.tddft.org/bmg/files/papers/85619.pdf, [31])
The calculation is very sensitive to the k-point grid, an extremely high number of k-points might be needed for convergence, especially for metals.

Tag: `compute_momentummatrix` (control.in)

Usage: `compute_momentummatrix` E_{min} E_{max} `k-point`

Purpose: Calculate and output the momentum matrix elements $\left\langle \psi_{n'\vec{k}} | \nabla_x | \psi_{n\vec{k}} \right\rangle$, $\left\langle \psi_{n'\vec{k}} | \nabla_y | \psi_{n\vec{k}} \right\rangle$, $\left\langle \psi_{n'\vec{k}} | \nabla_z | \psi_{n\vec{k}} \right\rangle$ for the k-point with the number `k-point` for KS-eigenstates that are within the energy window [E_{min},E_{max}] (in eV) relative to the internal zero. The output file is named *element_k_*`k-point`*.dat*. The unit is a_0^{-1} (bohr^{-1}).
If you are conducting a cluster calculation (no periodic boundary conditions) make sure to set `k-point` to 1.
Setting `k-point` to 0 will output the momentum matrix elements for all k-points into one container file (mommat.h5). To use this functionality, FHI-aims has to be compiled with the external hdf5 module (see Sec. H.2 for details).

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

Usage: `compute_dipolematrix` E_{min} E_{max} `k-point`

Purpose: Calculate and output the dipole matrix elements $\left\langle \psi_{n'\vec{k}} | x | \psi_{n\vec{k}} \right\rangle$, $\left\langle \psi_{n'\vec{k}} | y | \psi_{n\vec{k}} \right\rangle$, $\left\langle \psi_{n'\vec{k}} | z | \psi_{n\vec{k}} \right\rangle$ (position operator!) for the k-point with the number `k-point` for KS-eigenstates that are within the energy window [E_{min},E_{max}] (in eV) relative to the internal zero. The output file is named *element_k_*`k-point`*.dat*. The unit is a_0 (bohr).
If you are conducting a cluster calculation (no periodic boundary conditions) make sure to set `k-point` to 1.
Setting `k-point` to 0 will output the dipole matrix elements for all k-points into one container file (dipmat.h5). To use this functionality, FHI-aims has to be compiled with the external hdf5 module (see Sec. H.2 for details).

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

Usage: compute_dipolematrix_k_k E_{min} E_{max} k_k_method

Purpose: Calculate and output the dipole matrix elements $\left\langle \psi_{n'\vec{k'}} | x | \psi_{n\vec{k}} \right\rangle$, $\left\langle \psi_{n'\vec{k'}} | y | \psi_{n\vec{k}} \right\rangle$, $\left\langle \psi_{n'\vec{k'}} | z | \psi_{n\vec{k}} \right\rangle$ (position operator!) for all (k, k')-points for KS-eigenstates that are within the energy window [E_{min},E_{max}] (in eV) relative to the internal zero. The output file is named *dipmat_k.h5*. The unit is a_0 (bohr). With the option k_k_method (choises: 1, 2, 3) you can chose the method for calculating the matrix elements. *1* will first calculate the matrix elements (k, k')!!! for the atomic basis before transforming to the KS-basis (biggest array size: n_basis*n_k_points*n_k_points). *2* will calculate the matrix elements (k) for the atomic basis and transform to the KS-basis for all (k') (n_k_points times) (biggest array size: n_basis*n_k_points). *3* will calculate the matrix elements in the atomic basis and transform to the KS-basis for all (k, k') (n_k_points*n_k_points times) (biggest array size: n_basis). *1* needs the most memory, *3* does the most (repetitive) calculations.
The dipole matrix elements for all (k,k')-points are written into one container file (*dipmat_k.h5*). To use this functionality, FHI-aims has to be compiled with the external hdf5 module (see Sec. H.2 for details).

Tag: compute_kubo_greenwood (control.in)

Usage: compute_kubo_greenwood width FD_Temp E_{min} E_{max} ω_{min} ω_{max} n_ω i j

Purpose: Calculate and output the i j component (choices: (x, y or z) or (a a)) of the the Kubo-Greenwood transport properties optical conductivity $\sigma(\omega)$ and Seebeck coefficient S in units $(\Omega \cdot cm)^{-1}$ and $\mu V/K$, respectively. The (a a) choice triggers the calculation of the average of the three diagonal elements $L_{ij,xx}$, $\sigma_{ij,yy}$ and $\sigma_{ij,zz}$. As before, [E_{min},E_{max}] (in eV) determine the energy window (relative to the internal zero) within which the momentum-matrix elements are considered. The Delta-distribution is represented by a Gaussian function with width width (in eV) and the Fermi occupations are calculated at an electronic temperature FD_Temp (in eV). The spectra contain n_ω ω-values in the interval [ω_{min}, ω_{max}] (in eV). The output file is named *dielectric_i_j<Fermi-level>.out* and consists of the dielectric function, the optical conductivity and the Seebeck coefficient. Usually it is enough to include only a few states below and above the fermi level in the energy window [E_{min},E_{max}].
The calculation is very sensitive to the k-point grid, an extremely high number of k-points might be needed for convergence, especially for metals.

Tag: greenwood_method (control.in)

Usage: `greenwood_method method`

Purpose: Switches between the use of sparse or full transition matrices. (Choices: *sparse* or *full*). As of now, the keyword has to be set in case the `compute_kubo_greenwood` is used. The use of *sparse* matrices is recommended. (Keyword will be deprecated in the near future)[1.0ex]

3.35 Electronic Transport

Electronic transport in FHI-aims can be computed either using the built-in routine described in this chapter or with the help of aitranss-package (see Chapter 5.1). The built-in routine computes the electronic transport through a nanostructure connected to two, three or four semi-infinite leads using the Landauer-Büttiker formalism. Only calculations within the zero-bias limit are supported. All transport related actions are initiated with the keyword `transport` that is followed by the action required.

A typical work-flow of a transport calculation is as follows

1. Information for the semi-infinite leads are created. To this end the action `lead_calculation` should be used. Separate periodic calculation is required for each lead and the third lattice vector must point away from the nanostructure, i.e., into the semi-infinite lead. The region used to model the semi-infinite lead should be large enough so that the basis functions from the nanostructure region do not extend beyond the lead region.
2. Once all the leads are calculated transport through the nanostructure is calculated using the action `transport_calculation`. This produces the tunneling information through the nanostructure for each pair of the leads. The nanostructure should be large enough so that the basis functions from different leads do not overlap. In this calculation the lead atoms need to be included in the file `geometry.in` and their positions must be exactly the same as in the calculation for the lead information. In particular, they should not be relaxed when the geometry of the nanostructure is optimized. This calculation should also be run as a periodic calculation using a sufficient amount of k-points. The transport part of the calculation is done using only one k-point but in order to ensure converged density and potential of the nanostructure region more k-points are usually needed.

Tags for general section of `control.in`:

Tag: `transport` (control.in)

Usage: `transport action [further options]`

Purpose: This is the keyword that is used to control the built-in transport routines.

`action` is a string that specifies the kind of requested action in the transport calculation; any further needed options depend on `action`.

Specific actions `transport` keyword:

`transport` **sub-tag:** `lead_calculation` (control.in)

Usage: transport lead_calculation

Purpose: Computes the lead information for the given lead. Only one lead can be calculated at at time and each lead needs to be calculated separately. The calculation is periodic and the third lattice vector must point into the lead.

---

transport sub-tag: transport_calculation (control.in)

Usage: transport transport_calculation

Purpose: Performs the transport calculation. Using this keyword requires that the information for all the semi-infinite leads is already calculated.

---

transport sub-tag: tunneling_file_name (control.in)

Usage: transport tunneling_file_name filename

Purpose: Specifies the name of the file where the tunneling information is written. Each pair of the leads is written to a separate column in plain text format.

---

transport sub-tag: energy_range (control.in)

Usage: transport energy_range E_{min} E_{max} n_{steps}

Purpose: Sets the energy range for which the tunneling information is calculated. The energy range $E_{\mathrm{max}} - E_{\mathrm{min}}$ is divided into n_{steps} steps and the tunneling information is calculated for each step. The energy zero is referenced at the chemical potential of the nanostructure.

---

transport sub-tag: lead_i (control.in)

Usage: transport lead_i atom_index filename

Purpose: Tells the transport calculation where the information on the semi-infinite leads is stored. One line for each lead is required and the index i is substituted by the lead index (1, 2, 3, or 4). The option atom_index is set to the index of the first atom of the lead in question in the file geometry.in and the coordinates of the lead atoms must be the same used when calculating the lead information. The option filename provides the name of the file where the lead information is stored. Note that for a successful transport calculation information for at least two leads is needed. Hence, this keyword must be invoked at least twice. The maximum number of leads supported is currently four.

The Green's function for the semi-infinite leads needs to be solved iteratively for each energy step. The iteration is regularized adding a small complex part to the energy of the step being computed. At each iteration the magnitude of the regularizing part is decreased until either convergence of lower bound of the parameter is reached. The iteration can be controlled by several keywords.

---

transport sub-tag: number_of_boundary_iterations (control.in)

Usage: transport number_of_boundary_iterations number

Purpose: Sets the maximum number of iterations to solve the Green's function for each lead. Default: 10

---

transport sub-tag: boundary_treshold (control.in)

Usage: transport boundary_treshold number

Purpose: Sets the convergence criterion for the iteration of the Green's functions. The metric used is the maximum absolute value change in the matrices for the Green's functions of the leads. Default: 1.0

---

transport sub-tag: boundary_mix (control.in)

Usage: transport boundary_mix number

Purpose: Mixing value for the linear mixer in the iteration of the Green's function. Values below one correspond to under-relaxation. Default: 0.7

---

transport sub-tag: epsilon_start (control.in)

Usage: transport epsilon_start number

Purpose: Starting value for the regularizing complex parameter in the iteration for the Green's function for the semi-infinite lead. Only the magnitude of the parameter should be given with this keyword. Default: $0.02\imath$

---

transport sub-tag: epsilon_end (control.in)

Usage: transport epsilon_end number

Purpose: Ending value for the regularizing complex parameter in the iteration for the Green's function for the semi-infinite lead. Only the magnitude of the parameter should be given with this keyword. Default: $0.0001\imath$

---

transport sub-tag: fermi_level_fix (control.in)

Usage: `transport fermi_level_fix`

Purpose: Set the potential reference of the leads to their Fermi level. Since in the transport calculations the leads are calculated first and only after that incorporated into the system there is a question of a potential reference. Currently two options are available. The default option is to use the average of the energy levels of the lowest orbitals of the lead atoms. The second option invoked by this keyword is to align the Fermi levels of the leads between the lead calculations and the transport calculation.
Note that since the calculation of the lead is separate from the transport calculation in general the distance from the energy of the lowest lying orbital to the Fermi level of the calculation is not necessarily the same. This means that the leads in the lead calculation are not in the same environment as in the transport calculation and bringing them together introduces an alignment problem for the potential. From the transport calculation point of view this implies that a gate voltage is introduced into the system. In the transport calculation the value of this gate voltage is printed. In the tunneling results the energy is always referenced to the Fermi level of the transport calculation. Default: `.false.`

3.36 ESP charges

This sections describes how to calculate partial charges by fitting to the electro static potential (ESP). These charges are widely used in the context of force fields ([169], [52], [214], [25], CHELP: [50], CHELPG: [34], RESP-charges: [17], CHELP-BOW/CHELMO: [213], ESP-charge from multi-pole-moments: [111]). FHI-aims implements a simple method for cluster calculation (molecules) as well as two methods for solids (periodic boundary conditions) [39], [49].
The starting point for these methods is the calculation of the electro static potential at a sufficiently high number of grid points outside the vdw radius of the atoms. To define a space region for the grid two parameters are necessary: a minimal radius and a maximal radius around the atoms. These radii are defined as multiples of the vdw-radius of the atoms, see figure 3.18 for details. The values for the vdw radii of most atoms in the periodic table have been taken from "http://de.wikipedia.org/wiki/Van-der-Waals-Radius" ([30], [205], [160]). For the generation of the points cubic grids are used. For cluster calculations points within a cube encapsulating the spheres with the maximal radius (multiple of the vdw radius) around all atoms are generated. For periodic boundary conditions the provided unitcell is used. The points within the superposition of the spheres with the minimal radius (minimal multiple of the vdw radius) are excluded. The atom-centered radial grids are also available, mainly for test purposes. Here, all points lie on N spheres with radii between the minimal and maximal multiple of the vdw radius. The spacing between the radii of the N spheres is equidistant or logarithmic.
For the cluster case the function to fit to is a sum of Coulomb potentials with charges

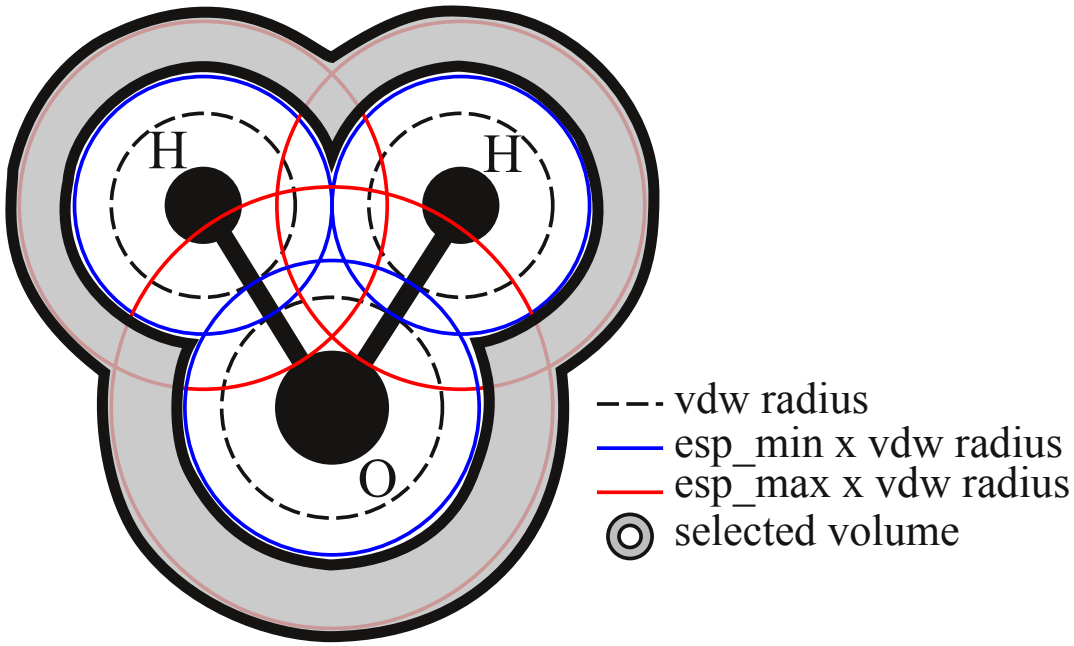


Figure 3.18: Definition of the volume used for the creation of grid points at which the potential is evaluated.

q_i, the ESP-charges, at the atomic position $\mathbf{R}_i$:

$$V_{ESP}(\mathbf{r}) = \sum_{i=1}^{N_{at}} \frac{q_i}{|\mathbf{r} - \mathbf{R}_i|} \tag{3.141}$$

The q_i are calculated by a least squares fit with the additional constraint of constant total charge $q_{tot} = \sum_{i=1}^{N_{at}} q_i$. We use the method of Lagrange multipliers to minimize the function:

$$F = \sum_{k=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_k}) - V_{ESP}(\mathbf{r_k})\right)^2 - \lambda \left(q_{tot} - \sum_{i=1}^{N_{at}} q_i\right)^2. \tag{3.142}$$

This can be translated into a system of linear equations:

$$\hat{\mathbf{A}}\mathbf{q} = \mathbf{B}. \tag{3.143}$$

with the N_{at+1} x N_{at+1} matrix $\hat{\mathbf{A}}$:

$$A_{ij} = \sum_{k=1}^{N_{grid}} \frac{1}{|\mathbf{r_k} - \mathbf{R}_i|} \frac{1}{|\mathbf{r_k} - \mathbf{R}_j|} \quad \text{with} \quad i,j \leq N_{at} \tag{3.144}$$

$$A_{i=N_{at}+1,j} = A_{i,j=N_{at}+1} = 1; \qquad A_{i=N_{at}+1,j=N_{at}+1} = 0$$

and the $N_{at} + 1$ vector $\mathbf{B}$:

$$B_i = \sum_{k=1}^{N_{grid}} \frac{V_{DFT}(\mathbf{r_k})}{|\mathbf{r_k} - \mathbf{R}_i|} \quad \text{with} \quad i \leq N_{at} \tag{3.145}$$

$$B_{i=N_{at}+1} = q_{tot} \tag{3.146}$$

$\mathbf{q}$ are the N_{at} charges.
For solids (periodic boundary conditions) the situation is more complicated because all charges are repeated infinitely and the potential is only defined up-to an arbitrary offset. The methods to solve this problem are based on Ewald summation. Details on method 1 can be found here [39] and on method two here [49]. The function for the potential generated by the ESP charges centered at the atoms of the unit-cell now reads:

$$V_{ESP}(\mathbf{r}) = \sum_{i=1,\mathbf{T}}^{N_{at}} q_i \frac{\mathrm{erfc}(\alpha|\mathbf{r} - \mathbf{R_{i,T}}|)}{|\mathbf{r} - \mathbf{R_{i,T}}|} + \frac{4\pi}{V_{cell}} \sum_{i=1,\mathbf{k}}^{N_{at}} q_i \cos(\mathbf{k}(\mathbf{r} - \mathbf{R}_i)) \frac{\exp^{-\frac{k^2}{4\alpha^2}}}{k^2} \tag{3.147}$$

with $\mathbf{T} = n_1\mathbf{a}_1 + n_2\mathbf{a}_2 + n_3\mathbf{a}_3$ mapping the lattice positions. $\mathbf{a_i}$ the lattice vectors, $n_i \in \mathbb{Z}$. $\mathbf{k} = m_1\mathbf{b}_1 + m_2\mathbf{b}_2 + m_3\mathbf{b}_3$ mapping the reciprocal space. $\mathbf{b_i}$ the reciprocal lattice vectors, $m_i \in \mathbb{Z}$. V_{cell} is the volume of the unit-cell. The parameter α is defined as $\alpha = \frac{\sqrt{\pi}}{R_c}$ with R_c the cutoff radius of the Ewald summation. For the second method Wolf summation is implemented as well:

$$V_{ESP}(\mathbf{r}) = \sum_{i=1}^{N_{at}} q_i \left[\frac{\mathrm{erfc}(\sqrt{\alpha}|\mathbf{r} - \mathbf{R_i}|)}{|\mathbf{r} - \mathbf{R_i}|} - \frac{\mathrm{erfc}(\sqrt{\alpha}R_c)}{R_c} + \left(\frac{\mathrm{erfc}(\sqrt{\alpha}R_c)}{R_c^2} + \frac{2\sqrt{\alpha}}{\sqrt{\pi}} \frac{\exp(\alpha R_c^2)}{R_c}\right)\right] \tag{3.148}$$

$$\mathrm{x}(|\mathbf{r} - \mathbf{R_i}| - R_c)$$

The function to minimize for method 1 is:

$$F_1^{PBC} = \sum_{k=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_k}) - V_{ESP}(\mathbf{r_k}) + \frac{1}{N_{grid}} \sum_{j=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_j}) - V_{ESP}(\mathbf{r_j}) \right) \right)^2 - \lambda \left(q_{tot} - \sum_{i=1}^{N_{at}} q_i \right) + \sum_{i=1}^{N_{at}} w_i \left(E_i^0 + \chi_i q_i + \frac{1}{2} J_i^{00} q_i^2 \right). \tag{3.149}$$

The χ_i and J_i^{00} are electro-negativity and self-Coulomb interaction of the respective elements, which can be used to constrain the desired charges. w_i are weighting factors for the constraints. This results in $\hat{A}$ and $\mathbf{B}$ for the linear equations system:

$$A_{ij} = \sum_{k=1}^{N_{grid}} \left(\frac{\partial V_{ESP}(\mathbf{r_k})}{\partial q_i} - \frac{1}{N_{grid}} \sum_{j=1}^{N_{grid}} \frac{\partial V_{ESP}(\mathbf{r_j})}{\partial q_i} \right) \text{ x } \left(\frac{\partial V_{ESP}(\mathbf{r_k})}{\partial q_j} - \frac{1}{N_{grid}} \sum_{m=1}^{N_{grid}} \frac{\partial V_{ESP}(\mathbf{r_m})}{\partial q_j} \right) + \frac{w_i}{2} J_i^{00} \delta_{ij} \quad ; \quad i,j \leq N_{at} \tag{3.150}$$

$$A_{i=N_{at}+1,j} = A_{i,j=N_{at}+1} = 1; \qquad A_{i=N_{at}+1,j=N_{at}+1} = 0$$

$$B_i = \sum_{k=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_k}) - \frac{1}{N_{grid}} \sum_{j=1}^{N_{grid}} V_{DFT}(\mathbf{r_j}) \right) \text{ x } \left(\frac{\partial V_{ESP}(\mathbf{r_k})}{\partial q_i} - \frac{1}{N_{grid}} \sum_{m=1}^{N_{grid}} \frac{\partial V_{ESP}(\mathbf{r_m})}{\partial q_i} \right) - w_m \frac{\chi_m}{2} \quad ; \quad i \leq N_{at}$$

$$B_{i=N_{at}+1} = q_{tot}$$

The function to minimize for method 2 is:

$$F_2^{PBC} = \sum_{k=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_k}) - \left(V_{ESP}(\mathbf{r_k}) + V_{DFT}^{offset} \right) \right)^2 - \lambda \left(q_{tot} - \sum_{i=1}^{N_{at}} q_i \right) + \beta \sum_{i=1}^{N_{at}} \left(q_i - q_{i0} \right)^2. \tag{3.151}$$

The constraint charges q_{i0} can be determined with other methods (e.g. Mulliken charge

analysis [171]), β is the weighing factor. This gives $\hat{A}$ and $\mathbf{B}$:

$$A_{ij} = \sum_{k=1}^{N_{grid}} \left(\frac{\partial V_{ESP}(\mathbf{r_k})}{\partial q_i} \frac{\partial V_{ESP}(\mathbf{r_k})}{\partial q_j} \right) + \beta \delta_{ij} \quad ; \quad i,j \leq N_{at} \tag{3.152}$$

$$A_{i=N_{at}+1,j} = A_{i,j=N_{at}+1} = \sum_{k=1}^{N_{grid}} \frac{\partial V_{ESP}(\mathbf{r_k})}{\partial q_j} \quad j \leq N_{at}$$

$$A_{i=N_{at}+2,j} = A_{i,j=N_{at}+2} = 1$$

$$A_{i=N_{at}+2,j=N_{at}+2} = A_{i=N_{at}+1,j=N_{at}+2} = A_{i=N_{at}+2,j=N_{at}+1} = 0$$

$$B_i = \sum_{k=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_k}) \frac{\partial V_{ESP}(\mathbf{r_k})}{\partial q_i} \right) - \beta q_{0i} \quad ; \quad i \leq N_{at}$$

$$B_{i=N_{at}+1} = \sum_{j=1}^{N_{grid}} V_{DFT}(\mathbf{r_j})$$

$$B_{i=N_{at}+2} = q_{tot}$$

Here the arbitrary offset of the potential V_{offset} is an additional fitting parameter, the matrix $\hat{A}$ is of dimension N_{at+2} x N_{at+2} and $\mathbf{B}$ of dimension N_{at+2}. For method 1 V_{offset} can be calculated as:

$$V_{offset} = \sum_{k=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_k}) - V_{ESP}(\mathbf{r_k}) \right) \tag{3.153}$$

from the fitted charges q_i. As a measure for the quality of the fit the root-mean-square ($RRMS$) is defined as:

$$RRMS = \left\{ \frac{\sum_{k=1}^{N_{grid}} \left((V_{ESP}(\mathbf{r_k}) + V_{offset}) - V_{DFT}(\mathbf{r_k}) \right)^2}{\sum_{k=1}^{N_{grid}} \left(V_{DFT}(\mathbf{r_k}) \right)^2} \right\}^2 \tag{3.154}$$

The current implementation is quite sensitive to the points chosen for the calculation of the electrostatic potential. Thorough studies regarding the parameters for the grid are advised! For periodic boundary conditions the ESP-charges calculated for transition densities are experimental, caution!! The ESP-charges from transition densities can be benchmarked against the dipole-moments calculated with `compute_dipolematrix`. Example for a periodic system:

```
output esp
esp n_radius 10
esp radius 1.0 2.
esp pbc_method 1
esp R_c 10
output esp
esp n_radius 10
esp radius 1.0 2.
esp pbc_method 2
esp R_c 30
output esp
```

```
esp radius 1.0 2.
esp pbc_method 2
esp R_c 20
esp equal_grid_n 10 10 10
```

The ESP-charges for the full potential will be calculated for a periodic system. At first with points within once the vdw radius and twice the vdw radius of the atoms, with 10 shells in between and a cutoff radius of 10Å for the Ewald summation, method 1 (Ewald summation) is used. Secondly with points within once the vdw radius and twice the vdw radius of the atoms, with 10 shells in between and a cutoff radius of 30Å, method 2 (Ewald summation) is used. Thirdly with points within once the vdw radius and twice the vdw radius of the atoms, with $10\times10\times10$ initial points on a cubic grid, method 2 is used.

Warning: For periodic boundary conditions the atoms in the supplied `geometry.in` must be within the first (central) Wigner-Seitz-cell. For a two layer system it would normally be possible to have one layer within the (central) Wigner-Seitz-cell and the second one sticking out at one side. Periodic boundary conditions will take care. The atoms sticking out would be shifted back into the (central) Wigner-Seitz-cell. This might lead to different ESP-charges and Dipole matrix elements. They are only invariant for collective translations of all atoms. The program will stop if a non-valid geometry is detected.

Tags for general section of `control.in`:

`output` **sub-tag:** `esp` (control.in)

Usage: `output esp`

Purpose: Calculate the ESP-charges at the atomic position from the full density with default settings. The default settings are to use a radial grid with equidistant radii and without mapping back to the unit-cell in case of periodic boundary conditions. The radial multipliers for the exclusion radius are 3 (Minimum) and 8 (Maximum) times the vdw-radius for the cluster case and 1 (Minimum) and 2 (Maximum) times the vdw-radius for pbc. In both cases 5 radial-shells are used. For pbc a cutoff radius for the Ewald summation of 10Å is used. The maximal multiples of the lattice vectors (r_m) and the reciprocal lattice vectors (k_m) used in the summation are 7 (for both). No constraints are applied to the charges.
The keyword can be used multiple times to calculate esp charges with different settings. The optional sub-sub keywords (see below) have to be set after `output esp` for every calculation separately, otherwise defaults are used.

Optional keywords for `output esp` :

- `esp spin` *spin*
  Select *spin* spin state (1 or 2) for the calculation of the esp charges from the transition density ρ_{ij} with states i -> j (default 1) at k-point *kpoint* (default 1).
- `esp state` *state_one state_two*
  Select states i=*state_one* -> j=*state_two* (i,j $\in$ [1,n_states]) for the calculation of the esp charges from the transition density ρ_{ij} with spin *spin* (default 1) at k-point *kpoint* (default 1).
- `esp kpoint` *k-point*
  Select states the `kpoint` (*k-point* $\in$ [1,n_k_points]) for the calculation of the esp charges from the transition density ρ_{ij} states i -> j (default 1) with spin *spin* (default 1).
- `esp radius` *min_radius max_radius*
  Select the minimal (min_radius) and maximal (max_radius) multiple of the vdw radius to select the volume where the potential is evaluated and the esp charges fitted. Defaults: 3 and 8 for clusters and 1 and 2 for pbc. *max_radius* should not be larger than smallest lattice vector for pbc and radial gird.
- `esp n_radius` *n_radius*
  Select the *n_radius* number of of radial shells that are used to create points in the selected volume. Default: 5
- `esp equal_grid_n` *n_grid_x n_grid_y n_grid_z*
  Select the number of points for the cubic grid in x - *n_grid_x*, y - *n_grid_y* and z - *n_grid_z* direction. Default: 10 10 10

- esp rm *r_m*
  PBC only. Maximum number *r_m* of multiples of the lattice vectors that are used in the real space sum of the Ewald summation. Default: 7

- esp km *k_m*
  PBC only. Maximum number *k_m* of multiples of the reciprocal lattice vectors that are used in the reciprocal space sum of the Ewald summation. Default: 7

- esp R_c *R_c*
  PBC only. Real space cutoff radius for the Ewald/Wolf-summation. Default: 20Å.

- esp pbc_method *method*
  PBC only. Switch between methods for periodic boundary conditions. Choices for *method*: *1* - method 1 with Ewald summation (3.150, 3.147); *2* - method 2 with Ewald summation (3.152, 3.147); *3* - method 3 with Wolf summation (3.152, 3.148). Default: 2

- esp grid *type*
  Switch between radial gird - 1, logarithmic radial grid - 2, cubic grid from lattice vectors - 3 and cubic grid from lattice vectors, but truncated in z-direction by maximal vdw-radius (exclude vacuum region for surface slabs) - 4. Default: 3.

- esp output_cube *type*
  Instead of fitting the ESP-charges to the potential, the potential is written to a cube-file (potential_esp_i.cube). A cubic grid is required. Switch between Hartree potential - 1, XC-potential - 2, X-potential - 3, C-potential - 4, Density - 5, and Hartree potential with the coordinates of voxels - 6.

- esp output_fit
  The ESP-charges are fitted to the potential and the potential calculated from the fitted ESP-charges is written to a cube-file (potential_esp_i.cube). A cubic grid is required.

- esp use_dip_for_cube
  The dipole correction is added to the cube output of the (fitted) potential. Only applied if a dipole correction was used during SCF.

Tag: **esp_constraint** (control.in)

Usage: esp_constraint method

Purpose: PBC only. Constrain the calculated ESP charges with periodic boundary conditions for method 1 or 2. The constraints will be used for all calculated ESP-charges. Due to technical reasons (array allocations) you have to specify the method you are using in *control.in* with this keyword. Choices for method are *1* and *2*. The actual constraints have to be specified for each atom in *geometry.in*.

Tag: **compute_esp_charges** (control.in)

Usage: compute_esp_charges E_{min} E_{max} min_vdw_radius grid_type max_vdw_radius n_radius k-point

Purpose: Calculate and output the ESP-charges for the transition states within the energy window [E_{min}, E_{max}] at k-point k-point. Data will be written to file *esp_element_*k-point*.dat*. The grid type grid_type (1 - radial, 2 - logarithmic radial, 3 - cubic, 4 - cubic from radii) will be used. The potential will be calculated on the n_radius radial shells within the volume created by min_vdw_radius and max_vdw_radius times the vdw radius of the atoms or in a cube with n_radiusxn_radiusxn_radius oints. The dipole moment calculated from the fitted charges q_i at atomic positions R_i is also written to file:

$$\mathbf{d_{ij}} = \sum_{k=1}^{N_{at}} q_k \mathbf{R_k} \tag{3.155}$$

for transition states $i \rightarrow j$.

Tags for geometry.in:

Tag: esp_constraint (geometry.in)

Usage: esp_constraint constraint_1 constraint_2 constraint_3

Purpose: PBC only. Define the constraints for the fit of the ESP-charges for each atom. Depending on the chosen method (esp_constraint method in *control.in*). Method 1 needs three parameters (χ, J^{00}, w) 3.150 and method 2 needs two parameters (q_0, β) 3.152 as defined above.

3.37 Magnetic Response

FHI-aims is capable of perturbatively calculating the response of the system to a magnetic field. The magnetic field can be either external or that associated with the magnetic moments of atomic nuclei. Currenty functionality includes:

- Nuclear magnetic shielding tensors and the spin-spin coupling tensors, which are the central parameters in nuclear magnetic resonance (NMR) spectroscopy;
- The magnetizability tensor, which is the proportionality between the induced magnetic moment and the magnetic field. It can be thought of as the magnetic analogue of the polarizability;
- Electric field gradient (experimental).

References:

- Martin Kaupp, Michael Bühl, and Vladimir G. Malkin, editors, *Calculation of NMR and EPR Parameters: Theory and Applications*, Wiley-VCH, Weinheim, Germany, 2004 — A comprehensive treatment of general theory and applications.
- V. Sychrovsky *et al.*, *Nuclear magnetic resonance spin-spin coupling constants from coupled perturbed density functional theory*, J. Chem. Phys. **113**, 3530 (2000) — Nonrelativistic J-couplings in DFT.
- G. Schreckenbach and T. Ziegler, *Calculation of NMR Shielding Tensors Using Gauge-Including Atomic Orbitals and Modern Density Functional Theory*, J. Phys. Chem. **99**, 606 (1995) — Standard theory of GIAO shieldings in DFT.
- T. Helgaker *et al.*, *Nuclear shielding constants by density functional theory with gauge including atomic orbitals*, J. Chem. Phys. **113**, 2983 (2000) — GIAO GGA correction (shieldings only).

The basic parameters of a magnetic response experiment are a collection of electron spins, $\boldsymbol{S} = \sum_i \boldsymbol{S}_i$, nuclear spins, $\boldsymbol{I} = \sum_i \boldsymbol{I}_i$, and an external magnetic field $\boldsymbol{B}$. Considering all possible couplings between these parameters, the fully general effective Hamiltonian has the form

$$H_S = H(\boldsymbol{S},\boldsymbol{S}) + H(\boldsymbol{S},\boldsymbol{I}) + H(\boldsymbol{S},\boldsymbol{B}) + H(\boldsymbol{I},\boldsymbol{I}) + H(\boldsymbol{I},\boldsymbol{B}) + H(\boldsymbol{B},\boldsymbol{B}). \quad (3.156)$$

We have not explicitly considered coupling to the electron orbital angular momentum, $\boldsymbol{L} = \sum_i \boldsymbol{L}_i$, whose effects are assumed to be incorporated into the electronic wavefunction. Based on which terms are dominant in a given spectroscopy, we can partition the Hamiltonian (3.156) as follows:

- The NMR nuclear spin Hamiltonian:

$$\begin{aligned} H^{\mathrm{NMR}} =& H(\boldsymbol{I},\boldsymbol{B}) + H(\boldsymbol{I},\boldsymbol{I}) \\ =& -\sum_A \boldsymbol{B}(\overleftrightarrow{1} - \overleftrightarrow{\sigma_A})\boldsymbol{\mu}_A + \sum_{A>B} h\boldsymbol{I}_A(\overleftrightarrow{D_{AB}} + \overleftrightarrow{J_{AB}})\boldsymbol{I}_B, \end{aligned} \quad (3.157)$$

where $\overleftrightarrow{\sigma}_A$ are the nuclear shielding tensors, $\overleftrightarrow{D_{AB}}$ and $\overleftrightarrow{J_{AB}}$ are the direct and indirect spin-spin coupling tensors, $\boldsymbol{I}_A$ is the nuclear spin of atom A, and $\boldsymbol{\mu}_A = \hbar\gamma_A\boldsymbol{I}_A$ is the corresponding nuclear magnetic dipole moment.

- The EPR spin Hamiltonian:

$$H^{\mathsf{EPR}} = H(\boldsymbol{S},\boldsymbol{S}) + H(\boldsymbol{S},\boldsymbol{I}) + H(\boldsymbol{S},\boldsymbol{B}). \tag{3.158}$$

- The effective Hamiltonian describing the second order response to an external magnetic field:

$$H^{\mathsf{Mag}} = -\frac{1}{2}\boldsymbol{B}\overleftrightarrow{\xi}\boldsymbol{B}, \tag{3.159}$$

where $\overleftrightarrow{\xi}$ is the magnetizability tensor. For a closed shell molecule, it describes the lowest order response of the energy to an external magnetic field.

The magnetic response parameters can be expressed as mixed second derivatives of the total electronic energy with respect to the external magnetic field and the magnetic moments,

$$\begin{aligned}
\overleftrightarrow{\sigma}_A &= \left.\frac{\partial^2 E(\boldsymbol{B},\boldsymbol{\mu})}{\partial\boldsymbol{B}\partial\boldsymbol{\mu}_A}\right|_{\boldsymbol{B}=0;\boldsymbol{\mu}=0}, \\
\overleftrightarrow{J_{AB}} &= \frac{\gamma_A\gamma_B}{2\pi}\left.\frac{\partial^2 E(\boldsymbol{B},\boldsymbol{\mu})}{\partial\boldsymbol{\mu}_A\partial\boldsymbol{\mu}_B}\right|_{\boldsymbol{B}=0;\boldsymbol{\mu}=0}, \\
\overleftrightarrow{\xi}_{AB} &= -\left.\frac{\partial^2 E(\boldsymbol{B},\boldsymbol{\mu})}{\partial^2\boldsymbol{B}}\right|_{\boldsymbol{B}=0;\boldsymbol{\mu}=0},
\end{aligned} \tag{3.160}$$

where $\boldsymbol{\mu}$ denotes the collection of all magnetic moments. In the first of Eqs. (3.160), the total energy should not contain the classical nuclear Zeeman term. The derivatives are, to a very good approximation, taken at zero values of $\boldsymbol{B}$ and $\boldsymbol{\mu}$, which allows us to consider only the linear response of the system.

In time-independent second-order perturbation theory, the general expression for a property $\overleftrightarrow{X}$ is

$$\overleftrightarrow{X} = \sum_i^{\mathsf{occ}} \left[\langle\psi_i|H^{MN}|\psi_i\rangle + 2\mathrm{Re}\,\langle\psi_i^M|H^N|\psi_i\rangle\right], \tag{3.161}$$

where $|\psi_i\rangle$ are the unperturbed wavefunctions, $|\psi^M\rangle$ are the first-order wavefunctions with respect to the perturbation M, and H^M and H^{MN} are the first and second derivatives of the Hamiltonian. The derivatives are calculated at zero values of the perturbation ($\boldsymbol{B}$ and/or $\boldsymbol{\mu}$). The total number of terms depends on the type of perturbation. For instance, in case of J-couplings, the paramagnetic part is a sum of three separate terms, $H^N = H^{\mathsf{FC}\boldsymbol{\mu}_A} + H^{\mathsf{PSO}\boldsymbol{\mu}_A} + H^{\mathsf{SD}\boldsymbol{\mu}_A}$, which are listed below.

Density functional perturbation theory

The first-order wavefunctions are calculated according to density functional perturbation theory (DFPT) (also called coupled-perturbed Kohn-Sham or coupled-perturbed Hartree-Fock). The DFPT steps are the following:

- Update the self-consistent first-order Hamiltonian:

$$H_\sigma^{(1)}(\boldsymbol{r}) = H_{\mathsf{NSC},\sigma}^{(1)}(\boldsymbol{r}) + \left[\frac{\partial v_{xc,\sigma}[n]}{\partial n_\sigma} n_\sigma^{(1)}(\boldsymbol{r}) + \frac{\partial v_{xc,\sigma}[n]}{\partial n_{\sigma'}} n_{\sigma'}^{(1)}(\boldsymbol{r})\right]_{n=n(\boldsymbol{r})} + H_{\mathsf{Ha}}^{(1)} n_\sigma(\boldsymbol{r}), \tag{3.162a}$$

  where $H_{\mathsf{NSC}}^{(1)}$ is the non-self-consistent part of the first-order Hamiltonian, $v_{xc,\sigma}$ is the xc-potential, $H_{\mathsf{Ha}}^{(1)}$ is the first-order Hartree potential, and $n_\sigma^{(1)}$ is the first-order density. First-order Hartree potential is computed only for an open-shell system and vanishes otherwise. For a closed shell system, only $n_\sigma^{(1)}$ in Eq. (3.162a) is updated, the rest need not be calculated more than once. In fact, the xc-kernel, which depends only on the unperturbed density, is the same for any perturbation and is calculated only once during the entire calculation. In Eq. (3.162a), only the LDA contribution is shown, but GGA and hybrids are also available. Note that for purely imaginary operators the perturbed density vanishes, which means that in LDA and GGA there is no contribution from the xc-kernel and it is sufficient to perform only a single DFPT step (this is the sum-over-states approach). With hybrids or Hartree-Fock, this does not hold because while the first-order density vanishes, the first-order density matrix does not.

- Compute the first-order wavefunctions:

$$\psi_n^{(1)}(\boldsymbol{r}) = \sum_i^{\mathsf{occ}} C_{in}^{(1)} \phi_i(\boldsymbol{r}) = \sum_i^{\mathsf{occ}} \phi_i(\boldsymbol{r}) \sum_k C_{ik} U_{kn}, \tag{3.162b}$$

  where $\phi_i(\boldsymbol{r})$ are the basis functions, $\boldsymbol{C}$ are the unperturbed wavefunction coefficients, $\boldsymbol{C}^{(1)}$ are the first-order coefficients, and

$$U_{mn} = -\frac{1}{2} \sum_{ij} C_{im} S_{ij}^{(1)} C_{jn} \tag{3.162c}$$

  if mn is an occupied-occupied or virtual-virtual pair, and

$$U_{mn} = \sum_{ij} \frac{C_{im} H_{ij}^{(1)} C_{jn} - C_{im} S_{ij}^{(1)} C_{jn} \epsilon_n}{\epsilon_n - \epsilon_m} \tag{3.162d}$$

  otherwise. $S^{(1)}$ is the first-order overlap matrix.

- Construct a new density matrix from the first-order changes in wavefunctions:

$$n_{ij\sigma}^{(1)} = \sum_{m\sigma}^{\mathsf{occ}} \langle \phi_i | \psi_{m\sigma} \rangle \langle \psi_{m\sigma}^{(1)} | \phi_j \rangle + \langle \phi_i | \psi_{m\sigma}^{(1)} \rangle \langle \psi_{m\sigma} | \phi_j \rangle = \sum_{m\sigma}^{\mathsf{occ}} c_{im} c_{jm}^{(1)*} + c_{im}^{(1)} c_{jm}. \tag{3.162e}$$

  The unperturbed coefficients are always real.

- Check convergence and, if necessary, perform density matrix mixing (default is to do Pulay mixing):

$$n_{ij\sigma}^{(1)} \to n_{ij\sigma}^{X,\mathsf{next}}. \tag{3.162f}$$

- Construct a new density from the density matrix:

$$n_\sigma^{(1)}(\boldsymbol{r}) = \sum_{ij}^{\mathsf{occ}} \phi_i(\boldsymbol{r}) n_{ij\sigma}^{(1)} \phi_j(\boldsymbol{r}). \tag{3.162g}$$

Nonrelativistic integrals

In the presence of an external magnetic field and NMR active nuclei (those with a nonzero magnetic moment) in the system, the magnetic vector potential takes the form

$$\boldsymbol{A} = \frac{1}{2}\boldsymbol{B} \times \boldsymbol{r} + \alpha^2 \sum_A \frac{\boldsymbol{\mu}_A \times \boldsymbol{r}_A}{r_A^3}, \tag{3.163}$$

where $\boldsymbol{r}_A = \boldsymbol{r} - \boldsymbol{R}_A$ is the position relative to atom A. Inserting this into the nonrelativistic Hamiltonian and taking the first and second derivatives with respect to the magnetic field and/or the nuclear magnetic moments, we arrive at the following terms:

- Orbital Zeeman:
$$H^{\boldsymbol{B}} = -\frac{i}{2}(\boldsymbol{r} \times \boldsymbol{\nabla}), \tag{3.164a}$$
- Spin Zeeman:
$$H^{\mathrm{SZ}\boldsymbol{B}} = \boldsymbol{S}, \tag{3.164b}$$
- Paramagnetic spin-orbit (PSO):
$$H^{\mathrm{PSO}\boldsymbol{\mu}_A} = -i\alpha^2 \frac{\boldsymbol{r}_A \times \boldsymbol{\nabla}}{r_A^3}, \tag{3.164c}$$
- Fermi contact (FC):
$$H^{\mathrm{FC}\boldsymbol{\mu}_A} = \frac{8\pi\alpha^2}{3}\delta(\boldsymbol{r}_A)\boldsymbol{S}, \tag{3.164d}$$
- Spin-dipole (SD):
$$H^{\mathrm{SD}\boldsymbol{\mu}_A} = \alpha^2 \left[\frac{3(\boldsymbol{S}\boldsymbol{r}_A)\boldsymbol{r}_A}{r_A^5} - \frac{\boldsymbol{S}}{r_A^3}\right], \tag{3.164e}$$
- Diamagnetic shielding (DS):
$$H^{\boldsymbol{B}\boldsymbol{\mu}_A} = \frac{\alpha^2}{2}\frac{\boldsymbol{r}\boldsymbol{r}_A - \boldsymbol{r}_A\boldsymbol{r}^T}{r_A^3}, \tag{3.164f}$$
- Diamagnetic spin-orbit (DSO):
$$H^{\boldsymbol{\mu}_A\boldsymbol{\mu}_B} = \frac{\alpha^4}{2}\frac{\boldsymbol{r}_A\boldsymbol{r}_B - \boldsymbol{r}_B\boldsymbol{r}_A^T}{r_A^3 r_B^3}, \tag{3.164g}$$
- Diamagnetic magnetizability:
$$H^{\boldsymbol{B}\boldsymbol{B}} = \frac{1}{4}(r^2 - \boldsymbol{r}\boldsymbol{r}^T). \tag{3.164h}$$

In case of shieldings and magnetizability, the GIAO formalism is used to overcome the slow convergence related to the nonuniqueness of the gauge origin. Each basis function ϕ_n is equipped with a phase factor of the form

$$\phi_n(\boldsymbol{r}) \to e^{-i\boldsymbol{A}_n\boldsymbol{r}}\phi_n(\boldsymbol{r}) = e^{-\frac{i}{2}(\boldsymbol{R}_n \times \boldsymbol{r})\boldsymbol{B}}\phi_n(\boldsymbol{r}), \tag{3.165}$$

where $\boldsymbol{A}_n = \frac{1}{2}\boldsymbol{B} \times (\boldsymbol{r} - \boldsymbol{R}_n)$ is the vector potential with the gauge origin shifted to the origin of basis function n, $\boldsymbol{R}_n$. This modifies some of the integrals (3.164) and leads to a few new ones:

- GIAO orbital Zeeman:

$$H_{mn}^{\mathrm{GIAO}\boldsymbol{B}} = \frac{i}{2}\left\langle \phi_m \,\middle|\, (\boldsymbol{R}_{mn}\times\boldsymbol{r})H_0 - \boldsymbol{r}_n\times\boldsymbol{\nabla}\,\middle|\,\phi_n\right\rangle + \langle\phi_m|V_{\mathrm{GGA},\alpha}^{\boldsymbol{B}}|\phi_n\rangle\,. \tag{3.166a}$$

- GIAO diamagnetic shielding:

$$H_{mn}^{\mathrm{GIAO}\boldsymbol{B}\boldsymbol{\mu}_A} = \frac{\alpha^2}{2}\left\langle \phi_m \,\middle|\, \frac{(\boldsymbol{R}_{mn}\times\boldsymbol{r})(\boldsymbol{r}_A\times\boldsymbol{\nabla})^T}{r_A^3}\,\middle|\,\phi_n\right\rangle. \tag{3.166b}$$

- GIAO diamagnetic magnetizability:

$$\begin{aligned} H_{mn}^{\mathrm{GIAO}\boldsymbol{B}\boldsymbol{B}} =& \frac{1}{4}\left\langle \phi_m \,\middle|\, 2(\boldsymbol{R}_{mn}\times\boldsymbol{r})(\boldsymbol{r}_n\times\boldsymbol{\nabla})^T \,\middle|\, \phi_n\right\rangle + \frac{1}{4}\left\langle \phi_m \,\middle|\, r_n^2 - \boldsymbol{r}_n\boldsymbol{r}_n^T \,\middle|\, \phi_n\right\rangle \\ & - \frac{1}{4}\left\langle \phi_m \,\middle|\, (\boldsymbol{R}_{mn}\times\boldsymbol{r})(\boldsymbol{R}_{mn}\times\boldsymbol{r})^T H_{\mathrm{LDA}} \,\middle|\, \phi_n\right\rangle + \left\langle \phi_m \,\middle|\, V_{\mathrm{GGA}}^{\boldsymbol{B}\boldsymbol{B}} \,\middle|\, \phi_n\right\rangle. \end{aligned} \tag{3.166c}$$

- First-order overlap matrix:

$$S_{mn}^{\mathrm{GIAO}\boldsymbol{B}} = \frac{i}{2}\left\langle \phi_m|\boldsymbol{R}_{mn}\times\boldsymbol{r}|\phi_n\right\rangle. \tag{3.166d}$$

- Second-order overlap matrix:

$$S_{mn}^{\mathrm{GIAO}\boldsymbol{B}\boldsymbol{B}} = -\frac{1}{4}\left\langle \phi_m|(\boldsymbol{R}_{mn}\times\boldsymbol{r})(\boldsymbol{R}_{mn}\times\boldsymbol{r})^T|\phi_n\right\rangle. \tag{3.166e}$$

In Eqs. (3.166), H_0 is the unperturbed DFT Hamiltonian, $V_{\mathrm{GGA}}^{\boldsymbol{B}}$ and $V_{\mathrm{GGA}}^{\boldsymbol{B}\boldsymbol{B}}$ are the first and second $\boldsymbol{B}$ derivatives of the xc-potential. In LDA, the latter terms vanish.

Table 3.4 illustrates which terms are required for calculating which properties.

Table 3.4: The number of different types of integrals required for J-couplings, shieldings, and magnetizabilities is 4. Without GIAOs, the number is 3 for shieldings and 2 for magnetizabilities.

	$\overleftrightarrow{J}_{AB}$				$\overleftrightarrow{\sigma}_A$ (no GIAO)		$\overleftrightarrow{\xi}$ (no GIAO)	
$\lvert\psi^M\rangle$	$H^{\mathrm{FC}\boldsymbol{\mu}_A}$	$H^{\mathrm{PSO}\boldsymbol{\mu}_A}$	$H^{\mathrm{SD}\boldsymbol{\mu}_A}$		$H^{\boldsymbol{B}}$		$H^{\boldsymbol{B}}$	
H^M	$H^{\mathrm{FC}\boldsymbol{\mu}_A}$	$H^{\mathrm{PSO}\boldsymbol{\mu}_A}$	$H^{\mathrm{SD}\boldsymbol{\mu}_A}$		$H^{\mathrm{PSO}\boldsymbol{\mu}_A}$		$H^{\boldsymbol{B}}$	
H^{MN}				$H^{\boldsymbol{\mu}_A\boldsymbol{\mu}_B}$		$H^{\boldsymbol{B}\boldsymbol{\mu}_B}$		$H^{\boldsymbol{B}\boldsymbol{B}}$

	$\overleftrightarrow{\sigma}_A$ (GIAO)			$\overleftrightarrow{\xi}$ (GIAO)		
$\lvert\psi^M\rangle$	$H^{\mathrm{GIAO}\boldsymbol{B}}, S^{\mathrm{GIAO}\boldsymbol{B}}$			$H^{\mathrm{GIAO}\boldsymbol{B}}, S^{\mathrm{GIAO}\boldsymbol{B}}$		
H^M		$H^{\mathrm{PSO}\boldsymbol{\mu}_A}$			$H^{\mathrm{GIAO}\boldsymbol{B}}$	
H^{MN}			$H^{\mathrm{GIAO}\boldsymbol{B}\boldsymbol{\mu}_A}$			$H^{\mathrm{GIAO}\boldsymbol{B}\boldsymbol{B}}, S^{\mathrm{GIAO}\boldsymbol{B}\boldsymbol{B}}$

Relativistic integrals

Following J. Autschbach and T. Ziegler, J. Chem. Phys. **113**, 936 (2000), the diamagnetic term is almost the same as in the non-relativistic case except for the ZORA factor $\mathcal{K}_{\mathsf{at}(j)}$:

$$\begin{aligned} \boldsymbol{H}^{\mathsf{ZORA},\boldsymbol{\mu}_A\boldsymbol{\mu}_B} =& \frac{\partial^2}{\partial\boldsymbol{\mu}_A\partial\boldsymbol{\mu}_B}\boldsymbol{A}\mathcal{K}_{\mathsf{at}}\boldsymbol{A} = 2\alpha^4\mathcal{K}_{\mathsf{at}}\frac{\boldsymbol{r}_A\boldsymbol{r}_B - \boldsymbol{r}_B\boldsymbol{r}_A^T}{r_A^3 r_B^3}, \\ \mathcal{K}_{\mathsf{at}(j)} =& \frac{c^2}{2c^2 - v_{\mathsf{at}(j)}^{\mathsf{free}}}. \end{aligned} \tag{3.167}$$

Note that because of $\mathcal{K}_{\mathsf{at}(j)}$ Eq. (3.167) needs to be explicitly symmetrized which is different from the non-relativistic case. The paramagnetic spin-orbit (PSO) term is similar to its non-relativistic equivalent,

$$\boldsymbol{H}^{\mathsf{ZORA},\mathsf{PSO}\boldsymbol{\mu}_A} = \frac{\partial}{\partial\boldsymbol{\mu}_A}\left(\boldsymbol{p}\mathcal{K}_{\mathsf{at}}\boldsymbol{A} + \boldsymbol{A}\mathcal{K}_{\mathsf{at}}\boldsymbol{p}\right) = -i\alpha^2\left(\mathcal{K}_{\mathsf{at}}\frac{\boldsymbol{r}_A\times\boldsymbol{\nabla}}{r_A^3} + \frac{\boldsymbol{r}_A\times\boldsymbol{\nabla}}{r_A^3}\mathcal{K}_{\mathsf{at}}\right), \tag{3.168}$$

but since the ZORA factor is orbital-dependent, it needs to be evaluated as two separate integrals,

$$\begin{aligned} &\left\langle\phi_m\middle|\,\boldsymbol{H}^{\mathsf{ZORA},\mathsf{PSO}\boldsymbol{\mu}_A}\middle|\,\phi_n\right\rangle \\ &= -i\alpha^2\boldsymbol{r}_A\times\left(\int\mathcal{K}_{\mathsf{at}(n)}\frac{\phi_m(\boldsymbol{r})\boldsymbol{\nabla}\phi_n(\boldsymbol{r})}{r_A^3}\mathrm{d}\,\boldsymbol{r} - \int\mathcal{K}_{\mathsf{at}(n)}\frac{\left[\boldsymbol{\nabla}\phi_m(\boldsymbol{r})\right]\phi_n(\boldsymbol{r})}{r_A^3}\mathrm{d}\,\boldsymbol{r}\right), \end{aligned} \tag{3.169}$$

where both terms need to be additionally symmetrized. The ZORA spin-orbit (ZSO) term is

$$\begin{aligned} H_j^{\mathsf{ZORA},\mathsf{ZSO}\boldsymbol{\mu}_A} =& \left[\frac{\partial}{\partial\boldsymbol{\mu}_A}i\boldsymbol{\sigma}\left(\boldsymbol{p}\mathcal{K}_{\mathsf{at}}\times\boldsymbol{A} + \boldsymbol{A}\mathcal{K}_{\mathsf{at}}\times\boldsymbol{p}\right)\right]_j \\ &= \alpha^2\left[\sigma_j\boldsymbol{\nabla}\left(\mathcal{K}\frac{\boldsymbol{r}_A}{r_A^3}\right) - \boldsymbol{\sigma}\nabla_j\left(\mathcal{K}\frac{\boldsymbol{r}_A}{r_A^3}\right)\right] \end{aligned} \tag{3.170}$$

in the j direction. The corresponding matrix elements are

$$\begin{aligned} \left\langle\phi_m\middle|\,H_j^{\mathsf{ZORA},\mathsf{ZSO}\boldsymbol{\mu}_A}\middle|\,\phi_n\right\rangle =& -\alpha^2\sigma_j\int\mathcal{K}_{\mathsf{at}(n)}\frac{\boldsymbol{r}_A}{r_A^3}\left[(\boldsymbol{\nabla}\phi_m(\boldsymbol{r}))\phi_n(\boldsymbol{r}) + \phi_m(\boldsymbol{r})\boldsymbol{\nabla}\phi_n(\boldsymbol{r})\right]\mathrm{d}\,\boldsymbol{r} \\ &+ \alpha^2\boldsymbol{\sigma}\int\mathcal{K}_{\mathsf{at}(n)}\frac{\boldsymbol{r}_A}{r_A^3}\left[(\nabla_j\phi_m(\boldsymbol{r}))\phi_n(\boldsymbol{r}) + \phi_m(\boldsymbol{r})\nabla_j\phi_n(\boldsymbol{r})\right]\mathrm{d}\,\boldsymbol{r}. \end{aligned} \tag{3.171}$$

If Eq. (3.171) were evaluated directly, the spin-orbitals would need to possess two independent complex components. However, we can decompose the ZSO operator as follows:

$$\begin{aligned} H_i^{\mathsf{ZORA},\mathsf{ZSO}\boldsymbol{\mu}_A} =& \alpha^2\sum_j\sigma_j H_{ij}^{\mathsf{ZORA},\mathsf{ZSO}\boldsymbol{\mu}_A}, \\ H_{ij}^{\mathsf{ZORA},\mathsf{ZSO}\boldsymbol{\mu}_A} =& \delta_{ij}\boldsymbol{\nabla}\left(\mathcal{K}\frac{\boldsymbol{r}_A}{r_A^3}\right) - \nabla_i\left(\mathcal{K}\frac{r_{Aj}}{r_A^3}\right) \end{aligned} \tag{3.172}$$

Each term $\sigma_j H_{ij}^{\mathsf{ZORA,ZSO}\boldsymbol{\mu}_A}$ is calculated separately by taking the spin quantization axis to be in the i direction in each perturbation calculation. This makes the ZSO operator diagonal in spin space and allows us to use spin-free real orbitals to represent the perturbations. After integrating by parts, each of those terms is expressed as

$$\begin{aligned}
\delta_{ij} \langle\phi_m|\boldsymbol{\nabla}\left(\mathcal{K}\frac{\boldsymbol{r}_A}{r_A^3}\right)|\phi_n\rangle + \langle\phi_m| - \nabla_i\left(\mathcal{K}\frac{r_{Aj}}{r_A^3}\right)|\phi_n\rangle = \\
- \int \mathcal{K}_{\mathsf{at}(n)}\frac{\boldsymbol{r}_A}{r_A^3}\left[(\boldsymbol{\nabla}\phi_m(\boldsymbol{r}))\phi_n(\boldsymbol{r}) + \phi_m(\boldsymbol{r})\boldsymbol{\nabla}\phi_n(\boldsymbol{r})\right]\mathrm{d}\,\boldsymbol{r} \\
+ \int \mathcal{K}_{\mathsf{at}(n)}\frac{r_{Aj}}{r_A^3}\left[(\nabla_i\phi_m(\boldsymbol{r}))\phi_n(\boldsymbol{r}) + \phi_m(\boldsymbol{r})\nabla_i\phi_n(\boldsymbol{r})\right]\mathrm{d}\,\boldsymbol{r},
\end{aligned} \tag{3.173}$$

where both terms need to be explicitly symmetrized because of $\mathcal{K}_{\mathsf{at}(n)}$. The observable (the J-coupling) is then computed as

$$\begin{aligned}
\overleftrightarrow{J}_{AB}^{\mathsf{ZSO}} &= \frac{\gamma_A\gamma_B}{2\pi}\sum_i^{\mathsf{occ}} 2\mathrm{Re}\,\langle\psi_i^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\boldsymbol{H}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\psi_i\rangle \\
&= \frac{\gamma_A\gamma_B}{2\pi}\sum_i^{\mathsf{occ}} 2\mathrm{Re}\Big(\langle\psi_{xx}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|H_{xx}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\psi_i\rangle + \langle\psi_{yy}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|H_{yy}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\psi_i\rangle + \langle\psi_{zz}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|H_{zz}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\psi_i\rangle \\
&+ 2\langle\psi_{xy}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|H_{xy}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\psi_i\rangle + 2\langle\psi_{xz}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|H_{xz}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\psi_i\rangle + 2\langle\psi_{yz}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|H_{yz}^{\mathsf{ZSO}\boldsymbol{\mu}_A}|\psi_i\rangle\Big).
\end{aligned} \tag{3.174}$$

Equation (3.174) along with the PSO and DSO terms forms the complete ZORA J-coupling.

Following R. Bouten *et al.*, J. Phys. Chem. A **104**, 5600 (2000), the ZORA shielding integrals are

$$\begin{aligned}
h_{mn}^{\mathsf{ZORA,GIAO}\boldsymbol{B}\boldsymbol{\mu}_A} &= \frac{\alpha^2}{2}\left\langle\phi_m\middle|\mathcal{K}_{\mathsf{at}(n)}\frac{(\boldsymbol{R}_{mn}\times\boldsymbol{r})(\boldsymbol{r}_A\times\boldsymbol{\nabla})^T + \boldsymbol{r}_A\boldsymbol{r}_n - \boldsymbol{r}_A\boldsymbol{r}_n^T}{r_A^3}\middle|\phi_n\right\rangle, \\
h_{mn}^{\mathsf{ZORA,GIAO}\boldsymbol{B}} &= \frac{i}{2}\langle\phi_m\,|\,(\boldsymbol{R}_{mn}\times\boldsymbol{r})H_{\mathsf{LDA}}\,|\,\phi_n\rangle + \left\langle\phi_m\middle|V_{\mathsf{GGA}}^{\boldsymbol{B}}\middle|\phi_n\right\rangle \\
&\quad - \frac{i}{2}\left\langle\phi_m\middle|\mathcal{K}_{\mathsf{at}(n)}\boldsymbol{r}_n\times\boldsymbol{\nabla}\middle|\phi_n\right\rangle, \\
h_{mn}^{\mathsf{ZORA,GIAO}\boldsymbol{\mu}_A} &= -i\alpha^2\left\langle\phi_m\middle|\mathcal{K}_{\mathsf{at}(n)}\frac{\boldsymbol{r}_A\times\boldsymbol{\nabla}}{r_A^3}\middle|\phi_n\right\rangle.
\end{aligned} \tag{3.175}$$

Dipolar couplings

The direct spin-spin coupling tensor, $\overleftrightarrow{D}_{AB}$, also called the dipolar coupling tensor, is expressed as

$$\overleftrightarrow{D}_{AB} = \frac{\hbar\alpha^2}{2\pi}\gamma_A\gamma_B\left(\frac{1}{R_{AB}^3} - \frac{3\boldsymbol{R}_{AB}\boldsymbol{R}_{AB}^T}{R_{AB}^5}\right), \tag{3.176}$$

where $\boldsymbol{R}_{AB}$ is the vector connecting atoms A and B. Because it does not depend on the electronic structure, its computation takes no time and it is always automatically included whenever a J-coupling calculation is requested

Current limitations

- Periodic systems not supported;
- Only the nonrelativistic formalism has been implemented (scalarrelativistic under development);
- Hybrid functionals not supported for shieldings and magnetizability;
- Meta functionals not supported.

For developers

Most of the magnetic response (MR) related source code resides in the directory `MagneticResponse`. The parent subroutine for doing MR calculations is `MR_main`. It sets up the environment depending on user input, calls `MR_core` which performs the actual calculations, and prints various information including the results of the calculation. `MR_core` is called separately for every type term that needs to be computed. If the first-order wavefunctions are required, a DFPT cycle is performed first. If a diamagnetic property is calculated, the DFPT part is skipped.

The module `integration` provides a subroutine that is used for all integrations. The only exception is the nonrelativistic FC term, which needs a completely separate subroutine because of the delta function. The bodies of most functions that are integrated over real space are found in `integrands.f90`. The result of each integration is written into a 2D block cyclic matrix (the `matrix_BC_out` argument of the `integrate`). Outside the integration subroutine, all linear algebra involving large arrays is done with Scalapack routines.

Example input

The following is a minimal example for running a magnetic response calculation.

`control.in`:

```
# H2O
xc                 pw-lda
magnetic_response  # Default is to calculate all magnetic
                   # response quantities
# Basis sets
...
```

`geometry.in`:

```
atom  0.00  0.00  0.00  O
```

```
magnetic_response
atom -0.96  0.00  0.00  H
magnetic_response
atom  0.32 -0.90  0.00  H
magnetic_response
```

Notes on performance

For best performance, it is recommended to always include the following flags in `control.in`:

```
load_balancing          .true.
use_local_index         .true.
collect_eigenvectors    .false.
```

In the timings section, the following terms are shown:

- "Fermi contact", "paramagnetic spin-orbit", "spin-dipole", "paramagnetic shielding", "paramagnetic magnetizability" — these are the integration times of the non-self-consistent parts of the Hamiltonian [$H^{(1)}_{\mathrm{NSC}}$ in Eq. (3.162a)].
- "diamagnetic spin-orbit", "diamagnetic shielding", "diamagnetic magnetizability" — integration of a diamagnetic magnetic property.
- "First-order xc" — integration of the first-order response of the xc-potential.
- "First-order density" — constructing the first-order electron density from the first-order density matrix (this is an integral over real space).
- "First-order Hartree" — integration of the first-order response of the Hartree potential.
- "First-order Ha update" — constructing the first-order Hartree potential from the first-order density. It mainly times calls to `update_hartree_potential_p1` and `sum_up_whole_potential_p1`.
- "Mat-mat multiplications" — total wall time spent on `pdgemm` and `pzgemm`.
- "packed <-> block cyclic" — total wall time spent on subroutines that do conversion from one matrix type to another. These are `packed_to_block_cyclic` and `block_cyclic_to_packed`.
- "Total" — total wall time spent on the magnetic response calculation. This is higher than the sum of the above, because it contains additional overhead from stuff that is in between the subroutines that are timed.
- "Total minus individual terms" — Difference between the total wall time and the sum of individual terms. If the total time is much higher than the sum of the individual terms then there is an unexpected bottleneck somewhere in the code that does not scale. Note that the difference is between the total time of the

slowest task and a sum where each term individually corresponds to that of the slowest task. Thus, do not be alarmed if this number becomes slightly negative (unlikely, but possible depending on how the load is balanced).

Maximum and minimum cpu times are shown where possible. In case the subroutine to be timed contains and explicit or implicit MPI barrier (e.g., `pdgemm`), individual cpu times are not meaningful and only the maximum wall time is shown.

Tags for general section of `control.in`:

Tag: `magnetic_response` (control.in)

Usage: `magnetic_response options`

Purpose: Primary keyword for doing magnetic response calculations. This is the minimum that is required in `control.in` and by default leads to the full calculation of J-couplings, shieldings, and the magnetizability.

This keyword can be followed by a number of options:

- `J_coupling` *or* `J` *or* `j` — Calculate J-couplings only.
- `shielding` *or* `s` — Calculate the shieldings only.
- `magnet` *or* `m` — Calculate the magnetizability only.
- `fc` — Calculate the FC contribution to J-couplings only.
- `po` — Calculate the PSO contribution to J-couplings only.
- `sd` — Calculate the SD contribution to J-couplings only.
- `do` — Calculate the DSO contribution to J-couplings only.
- `shielding_p` *or* `s_p` — Calculate the paramagnetic contribution to the shieldings only.
- `shield_d` *or* `s_d` — Calculate the diamagnetic contribution to the shieldings only.
- `magnet_p` *or* `m_p` — Calculate the paramagnetic contribution to the magnetizability only.
- `magnet_d` *or* `m_d` — Calculate the diamagnetic contribution to the magnetizability only.
- `no_giao` — Calculate the shieldings and magnetizability using the standard formalism without GIAOs (default: false, i.e., with GIAOs).
- `full` — Calculate the full tensors (default: only diagonal elements).

Comment: any combination of options works. For example,
`magnetic_response fc s po full`
calculates the FC and PSO terms of J-couplings and the shielding tensors including off-diagonal elements. The default is equivalent to
`magnetic_response J_coupling shielding magnet`

Tag: `dfpt_accuracy_n1` (control.in)

Usage: `dfpt_accuracy_n1 value`

Purpose: Convergence criterion for the DFPT self-consistency cycle, based on the RMS change in the first-order density matrix. Specifically, the unmixed output density matrix is checked against the input density matrix corresponding to the same iteration. The RMS value is calculated over all directions and spins that are being processed.

`value` is a real positive number (in electrons). Default: 1d-9.

Tag: `dfpt_iter_limit` (control.in)

Usage: `dfpt_iter_limit number`

Purpose: Maximum number of DFPT cycles.

`number` is an integer number. Default: 40.

Tag: `dfpt_linear_mix_param` (control.in)

Usage: `dfpt_linear_mix_param value`

Purpose: Parameter for linear mixing of first-order electron density. Used in Pulay and simple linear mixing.

`value` is a real number between 0 and 1. Default: 1.0

Tag: `dfpt_pulay_steps` (control.in)

Usage: `dfpt_pulay_steps number`

Purpose: Number of steps kept in memory for Pulay mixing. Value of 1 corresponds to simple linear mixing.

`number` is a positive integer number. Default: 8

Tag: `mr_gauge_origin` (control.in)

Usage: `mr_gauge_origin x y z`

Purpose: Gauge origin for the calculation of the shieldings or the magnetizability. This has no effect with GIAOs.

`x y z` are real numbers (in Å) that specify the position of the gauge origin. Default: the center of mass.

Comment: The center of mass is based on the natural abundance of elements. If this is not desirable, the user can set the gauge origin manually.

Tag: `output_sxml` (control.in)

Usage: `output_sxml` [name]

Purpose: If present, the spin system and the results of calculations are printed into an sxml file [Biternas *et al.*, J. Mag. Res. **240**, 124 (2014)], which can serve as input to other software, e.g., Spinach [spindynamics.org].

The name of the output file is `name` if present. Otherwise, the default name is `aims.xsml`.

Tag: `mr_experimental` (control.in)

Usage: `mr_experimental`

Purpose: Overrides any safety checks that would cause the run to stop. This allows the user to test features that are still in development and even avoid simple sanity checks such calculating the shieldings without specifying any atoms in geometry.in.

Tags for `geometry.in`:

Tag: `magnetic_response` (geometry.in)

Usage: `magnetic_response`

Purpose: Includes the current atom in the magnetic response calculations. If only the magnetizability is required, this keyword need not be used in `geometry.in`. Otherwise, the calculation of the shieldings or J-couplings is aborted if no atoms are flagged for MR calculations in `geometry.in`

Tag: `magnetic_moment` (geometry.in)

Usage: `magnetic_moment value`

Purpose: Overrides the default magnetic moment for the given atom. The default values (in units of the nuclear magneton μ_N) can be found in `MagneticResponse/MR_nuclear_data.f90`. In case of J-couplings, the isotopes used are also printed in the output.

`value`, a real number, is the magnetic moment in atomic units (-5.157d-4 for O-17).

Tag: `nuclear_spin` (geometry.in)

Usage: `nuclear_spin value`

Purpose: Overrides the default nuclear spin for the given atom. The default values can be found in `MagneticResponse/MR_nuclear_data.f90` and are also printed in the output for J-coupling calculations.

`value`, a real number, is the nuclear spin (2.5 for O-17).

Tag: `isotope` (geometry.in)

Usage: `isotope number`

Purpose: Overrides the default isotope mass number for the given atom. For more flexibility, the magnetic moment and spin can be specified separately with the above keywords. The default isotopes numbers can be found in `MagneticResponse/MR_nuclear_data.f90`.

`number`, a positive integer, is the mass of the given atom (17 for O-17).

3.38 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 `KS_method parallel` is used, keyword `collect_eigenvectors .false.` switches off the collection of the full eigenvectors c_{il} on each CPU, saving both communication time and a significant amount of memory. This is the default where possible but the default is currently switched silently between `.false.` (efficient) and `.true.` (inefficient) depending on the details of the calculation.

- 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 Kerker `preconditioner` (part of the density mixing step is switched on by default for all periodic calculations, but does not scale well to large processor counts and with system size. If the density mixing step costs a significant amount of time (see FHI-aims timings, printed at the end of each s.c.f. step in the output), consider switching off the Kerker preconditioner. For systems with a band gap, it is often not needed and the much cheaper default Pulay mixer will work as well.

- In the cluster case, keyword `use_density_matrix_hf` 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 `KS_method parallel` 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.
- 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 keyword, `use_alltoall` , that allows to switch the communication behaviour of FHI-aims for very large runs when parallel linear algebra (`KS_method parallel`) 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 `KS_method parallel` is used, allows to switch off the collection of all eigenvectors to each CPU, saving memory and computation time.

Restriction: Local copies of the eigenvectors are needed for some post-processing functionality.

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

For memory efficiency reasons, this keyword should be set to `.false.` whenever possible. Otherwise, especially the largest and most demanding calculations will run out of memory. When `.true.`, this keyword creates a complete and up-to-date copy of every eigenvector for the k-point assigned to it on every MPI task. When `.false.`, this information is communicated only when unavoidable. In principle, every operation in FHI-aims should be implemented so as to keep this keyword set to `.false.` and it is only human time that keeps us from doing it.

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 distribution of work across CPUs for real-space grid operations. When load balancing is enabled via this keyword, this performance hit 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.` (or if_scalapack, see below). Default: `.false.`

This feature, which was implemented by Rainer Johanni, eliminates the negative impact on performance incurred by the `use_local_index` keyword. It should always be used whenever memory becomes a bottleneck for a calculation. For more information about what this keyword does, please see the documentation for the `use_local_index` keyword, as the two keywords are closely intertwined.

Load balancing requires that `KS_method parallel` be used: that is, there is more than one CPU for cluster calculations or more CPUs than k-points for periodic calculations. If the `if_scalapack` option is supplied, load balancing will be turned on when `KS_method parallel` is being used and will be turned off when `KS_method serial` is being used.

When this keyword is set to `.true.` or `if_scalapack`, the `use_local_index` keyword will be set to the same value by default.

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 `KS_method parallel` is used.
- In the periodic case: for more than one k-point, and/or if `KS_method parallel` 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 `KS_method parallel`.

When running on a system with many CPUs, it can be much faster to use "all-to-all" communcation (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 at 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 "sendrcv" (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_mpi_in_place` (control.in)

Usage: use_mpi_in_place flag

Purpose: Allows some collective communication calls to be handled using the "MPI_IN_PLACE" flag of the MPI specification.

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

Only relevant for parallel runs.

When running on a system with many CPUs, it can be a bit more memory efficient to use "MPI_IN_PLACE" communication. Then, instead of separate send- and receive buffers, a single buffer is used and information is updated "in place".

However, not all MPI implementations seem to implement this feature correctly. Thus, this choice can sometimes lead to problems or even errors in the results, depending on the MPI library and version that was used.

The code checks for specific problems related to "MPI_IN_PLACE" and switches to use_mpi_in_place .false. if problems are encountered.

If anyone identifies further problems related to the use of "MPI_IN_PLACE", please 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 , GGA, and hybrid functionals using KS_method parallel and packed matrices, but not for some non-standard options.

flag is a logical string, either .false. or .true. (or if_scalapack, see below). Default: .false.

Originally, FHI-aims stored its real-space Hamiltonian/overlap matrices in a non-distributed fashion, i.e. every CPU has the full copies of the real-space Hamiltonian/overlap matrices. While this makes the math easier internally, it will lead to a memory bottleneck for large calculations: as the number of atoms in a calculation increases, the sizes of the real-space matrices increase, and eventually the real-space matrices are too large to fit in memory. Adding more CPUs to a calculation does not fix this problem, since each CPU has the full copies of the matrices.

For very large systems with many CPUs, it thus becomes necessary to spread the real-space matrices across CPUs. This is done using a method known as "domain decomposition" (or, informally, local indexing), where we assign batches of integration grid points close to one another to the same CPU. Each CPU then stores only the portions of the real-space Hamiltonian/overlap matrices which have non-zero support on the integration points which it possesses. The real-space matrices are thus distributed across CPUs, considerably decreasing their sizes. Should a calculation still suffer from memory

issues associated with the sizes of the real-space matrices, we can increase the number of CPUs to reduce the memory overhead on each individual CPU.

While the domain decomposition method will spread the real-space matrices across CPUs and eliminates the memory bottleneck previously mentioned, eventually we will need to solve the Kohn-Sham eigenvalue problem. To solve the Kohn-Sham eienvalue problem, we need to generate the Hamiltonian entering into the eigensolver from the real-space Hamiltonian, which is now distributed across CPUs. The subsequent merge of all results into the BLACS infrastructure used by the eigensolvers supported by FHI-aims then becomes more difficult, and some performance overhead results from the altered load on each CPU. This option is therefore not used by a standard call to FHI-aims, but must be switched on explicitly if needed.

To overcome the performance overhead associated with `use_local_index` keyword, it is *strongly* recommended that the user also try using the `load_balancing` keyword, which enables load balancing. Load balancing will eliminate the overhead associated with the `use_local_index` keyword, but it is not enabled for all non-standard functionality in FHI-aims, hence why we do not enable it by default.

Domain decomposition requires `KS_method parallel`: that is, there is more than one CPU for cluster calculations or more CPUs than k-points for periodic calculations. If the `if_scalapack` option is supplied, domain decomposition will be turned on when `KS_method parallel` is being used and will be turned off when `KS_method serial` is being used.

Domain composition also requires `prune_basis_once` and a parallel `grid_partitioning_method` . These are enabled by default for FHI-aims calculations, so you shouldn't worry about setting them yourself.

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.

3.39 Fragment molecular orbital DFT calculations

The fragment molecular orbital (FMO or FO) scheme allows the efficient calculation of transfer matrix elements for transport calculations. The two reference states are constructed from isolated calculations of the respective fragments (donor and acceptor). This circumvents the electron delocalistion error, but neglects any interactions between the fragments.

Important: This functionality is not yet available for periodic systems.
Important: The δ+-FODFT scheme is not optimized for large systems.

FODFT flavours

Depending on the approximations done to implement the FODFT method we distinguish between three different flavours of FODFT. Please see [209] for a detailed description and assessment of each. Within FHIaims it is possible to use all different FODFT-schemes.

$H^{2n}@DA$

This is the classic implementation by [211]. The fragment calculations are always done for the neutral fragments and the Hamiltonian is constructed using a wrong number of electrons. $H_{ab} = H_{ba}$ even for hetero dimers.

To use this flavour, simply calculate neutral fragments and continue with the default FODFT options.

$H^{2n-1}@DA$ / $H^{2n+1}@D^{-}A^{-}$

This implementation was first used within the CPMD code[177]. Here, neutral (double anionic) fragment calculations are combined for hole (electron) transfer with a reset of the occupation number in the highest occupied molecular orbital. This leads to the correct number of electrons for the construction of the Hamiltonian.

To use this flavour, calculate neutral (or anionic) fragments and use the `fo_flavour` keyword with the option `reset`.

$H^{2n\pm1}@D^{\pm}A$

This scheme was first implemented within FHIaims and uses charged fragment calculations. This results in slightly different approximations with improved accuracy in the electronic coupling values. Please see [209] for more details.

To use this flavour, calculate appropriately charged fragments and continue with the default FODFT options.

$\delta+$-FODFT

The implementation of the $\delta+$-FODFT scheme is not optimized for large systems and considered *experimental*. In our study[209] we found no significant improvement when using this scheme.

Theoretical Background

The following section intends to give a quick overview over the theory and the implementation within FHIaims. For detailed theory we refer to the before mentioned publications. The following theory is for the charged FODFT flavour ($H^{2n\pm1}@D^{\pm}A$).

In principle, the transfer matrix element for hole transport between the HOMO and the LUMO of two molecules is given by

$$H_{AB} = \langle \Psi^D_{\text{HOMO}} | \hat{H}_{KS} | \Psi^A_{LUMO} \rangle. \tag{3.177}$$

Obtain the fragment wavefunctions

First, two standard-DFT calculations are done to obtain the fragment wavefunctions. Some additional output is needed for the subsequent combination of the fragments to the full system.

Combine the fragment wavefunctions and extract the transfer matrix element

The saved wavefunctions of both fragment calculations are used to obtain the full, non-interacting electron density of the combined system (D+A). Although FHIaims uses atom centered basis functions, it is not generally possible to re-use any restart file. If the ordering of the `atoms` in the *geometry.in* files is identical, it is possible to use a restart file for any geometry with translational symmetry. If the geometry is rotated, it is necessary to rotate the wave function.

For this non self-consistent density the Hamiltonian is calculated and used to determine H_{AB}. We therefore use the non-interacting density, but the interacting Hamiltonian with the correct number of electrons. This is in contrast to other implementations, where only the Hamiltonian of the neutral fragment is used.

The first step is a Loewdin-Orthogonalization of the Kohn-Sham-eigenvector.

$$\mathbf{U}^\top \mathbf{S} \mathbf{U} = \mathbf{s} \tag{3.178}$$

$$\mathbf{X} = \mathbf{S}^{-\frac{1}{2}} = \mathbf{U}\mathbf{s}^{-\frac{1}{2}}\mathbf{U}^\top \tag{3.179}$$

$$\begin{aligned} \mathbf{C}' &= \mathbf{X}^{-1}\mathbf{C} \\ \mathbf{H}' &= \mathbf{X}^\top \mathbf{H} \mathbf{X} \end{aligned} \tag{3.180}$$

In the next step, the Hamiltonian is transformed,

$$\mathbf{H'}_{\text{st.}} = \mathbf{C'}^{\top}\mathbf{H'}\mathbf{C}, \tag{3.181}$$

and the overlap of states is calculated:

$$\mathbf{S}_{\text{st.}} = \mathbf{C}^{\top} \cdot \mathbf{C}. \tag{3.182}$$

In the final step, the transfer matrix element H_{AB} is calculated with

$$H_{AB} = \left(1 - \mathbf{S}_{\text{st.}}(\text{a,b})^2\right)^{-1} \left[\mathbf{H'}_{\text{st.}}(\text{a,b}) - \mathbf{S}_{\text{st.}}(\text{a,b}) \left(\mathbf{H'}_{\text{st.}}(\text{a,a}) + \mathbf{H'}_{\text{st.}}(\text{b,b})\right)/2\right] \tag{3.183}$$

Important technical hints

When creating the input geometries for the calculations, the `geometry.in` must mirror the actual fragmentation. This means, the order of atoms has to be consistent for the dimer and the fragments.

Example: Calculating the H_{AB} for two methane molecules (CH_4^+ – CH_4)

Complete System, CH_4^+–CH_4

```
#fragment 1
atom     12.7490 4.3034  0.0000  C
atom     13.8413 4.3037  0.0000  H
atom     12.3850 4.7804  -0.9128 H
atom     12.3850 4.8558  0.8693  H
atom     12.3851 3.2750  0.0435  H
#fragment 2
atom     -2.6946 4.4740  0.0002  C
atom     -1.6023 4.4740  0.0000  H
atom     -3.0586 3.6144  -0.5671 H
atom     -3.0586 5.3949  -0.4609 H
atom     -3.0585 4.4127  1.0278  H
```

Fragment 1, CH_4^+

```
atom     12.7490 4.3034  0.0000  C
atom     13.8413 4.3037  0.0000  H
atom     12.3850 4.7804  -0.9128 H
atom     12.3850 4.8558  0.8693  H
atom     12.3851 3.2750  0.0435  H
```

Fragment 2, CH_4

```
atom     -2.6946 4.4740  0.0002  C
atom     -1.6023 4.4740  0.0000  H
atom     -3.0586 3.6144  -0.5671 H
atom     -3.0586 5.3949  -0.4609 H
atom     -3.0585 4.4127  1.0278  H
```

Folder structure for FODFT-calculation

At least three separate calculations (fragment1, fragment2 and combination step) are needed for a complete FODFT run. To maintain a clean structure and avoid copying of files, a special scheme is enforced. The folders can have arbitrary names, but need to be in the same root directory. This allows easy reuse of fragment calculations for different dimer calculations.

```
- fodft\_something (main calculation folder)
|
+-- dimer_01
    |
    +-- control.in
    +-- geometry.in
+-- frag1/
    |
    +-- control.in
    +-- geometry.in
+-- frag2/
    |
    +-- control.in
    +-- geometry.in
```

δ^+ FO-DFT - Inclusion of the local potential

In our implementation, an additional option is available to include interactions between the fragments by means of an embedding scheme, using the full local potential (V_{loc}) of the embedded fragment. In contrast to the standard implementation, the fragment wavefunction Ψ_A is polarized by the potential of the second fragment Ψ_B during the SCF cycle.

The general procedure is:

1. Calculation of fragment wavefunctions
   (a) Calculate Fragment A, export potential A
   (b) Import potential A, calculate Fragment B, export new potential B
   (c) Import potential B, calculate Fragment A**
2. Combination of fragment wavefunctions
3. Determination of H_{AB}

**: Please note that this step might not be necessary if fragment A was choosen wisely, depending on the polarizability of the fragments.

The embedding can be used to iteratively converge towards a final solution, but as usual the first steps are the most important ones for the polarization.

IMPORTANT: Since this method requires the reassignment of grid points from the fragment calculations for the local potential, the current implementation is not optimized towards large systems.

Tags for general section of `control.in`:

Tag: `fo_dft` (control.in)

Usage: `fo_dft type`

Purpose: This is the central control keyword for fragment orbital DFT. Its main purpose is the control of the desired calculation step (fragment or combined system) and associated choices. Depending on `type`, further flags or lines are necessary. Those are explained below.

Options: Possible options are `fragment` and `final`.

`fo_dft` **sub-tag:** `fragment` (control.in)

Usage: `fo_dft fragment`

Purpose: This keyword indicates that a fragment for FODFT will be calculated. The restart information (`restart.frag`) and an additional output file (`info.frag`) with information for the recombination of the fragments for the final FODFT run will be written.

`fo_dft` **sub-tag:** `final` (control.in)

Usage: `fo_dft final`

Purpose: This keyword indicates that this is the final FODFT run, where to wavefunctions of both fragments are combined and the transfer matrix element is calculated.
IMPORTANT: This calculation can only be started *after* both fragments have been calculated!

Tag: `fo_orbitals` (control.in)

Usage: `fo_orbitals state1 state2 range1 range2 type`

Required input for the final FODFT step. Determines the actual states of interest for the calculation of the coupling elements.

- `state1`, `state2`: [Integer] – Isolated fragment states for the determination of the respective matrix element.
- `range1`, `range2`: [Integer] – Default 1. Gives a range for the selected states.

  (state1 $= 5$ and range1 $= 2$ will include state 5 and state 6 of fragment 1.)
- `type`: [String] Determines whether the matrix element for the spin-up (`up` or `elec`) or spin-down (`dn` or `hole`) Hamiltonian is calculated. In most cases, spin down means hole transport (excess hole on fragment 1 to fragment 2) and spin up means electron transport (excess electron on fragment 1 to fragment 2).

  Example Zn^+-Zn: `fo_orbitals 15 15 1 1 hole`
  will output the transfer matrix elements for hole transfer between the LUMO of Zn^+ (fragment 1) and the HOMO of Zn (fragment 2).

If not deactivated via `fo_verbosity 0`, the full H_{ab} submatrix will be written to the file `full_hab_submatrix` and any matrix element can be retrieved from this file after the calculation.

Tag: `fo_folders` (control.in)

OPTIONAL Usage: `fo_folders folder_frag1 folder_frag2`

Purpose: This keyword allows the specification of custom names for the folders with the fragment calculation. If not set, standard names (`frag1_00` and `frag2_00`) are used.

Tag: `fo_flavour` (control.in)

OPTIONAL Usage: `fo_flavour flavour`

Purpose: This keyword allows the specification of the FO-DFT flavour to be used for the calculation. See FODFT theory section for details. Options:

- `default` - no occupations are modified. For $H^{2n}@DA$ and $H^{2n\pm1}@D^{\pm}A$ flavours. This is the default.
- `reset` - selects the $H^{2n-1}@DA/H^{2n+1}@D^{-}A^{-}$ scheme.

Note: There is no option yet to calculate H_{ab} and H_{ba} within one FHIaims run. If the system is heterogeneous, two dimer steps with appropriate fragment ordering and charges are necessary!

Tag: `fo_verbosity` (control.in)

Usage: `fo_verbosity level`

Purpose: Control the verbosity of the FO-DFT output. Options:

- 0: Write minimal output to output file. No retrieval of arbitrary couplings without restarting the dimer calculation possible.
- 1: Write calculated couplings to output file and the full H_{ab} matrix to the file `full_hab_submatrix`. This is the default.
- 2: Write all elements for selected states from eq. 3.183 ($S_{st}, H'_{st}, \ldots$) to aims output.

Tag: `fo_deltaplus` (control.in)

Usage: `fo_deltaplus .true.`

Purpose: Enables the polarized δ^+ fragment calculations.

With this keyword, the full local potential of the actual fragment (Hartree + nuclei) will be written to files (`output_potential_0000*`). In addition, if previous potential files are found within the calculation folder, they will be read in and added as an external potential to the current calculation.

Important: In order to use this improved scheme, the (empty) atom positions of the other fragment have to be included in the `geometry.in` (see 3.39).

Important: The number of cores (determined by mpirun -n or similar) has to be the same for both fragments, or the calculation will fail!

Note: Due to differences in the partitioning of the FHIaims grid between these runs it is necessary to reassign the potential to each grid point. Depending on the system size and the number of MPI processes (communication!), this will take some time.

Additional tags for polarized FODFT in `geometry.in` and `control.in`:

The following changes are only necessary for polarized FODFT (`fo_deltaplus`). In order to allow a polarization of the fragment wavefunction, the atom centered integration grids have to be extended to the region of the embedded fragment.

geometry.in:

The atomic positions of the second, embedded fragment must be included as empty sites with the tag `empty` (instead of `atom`). In addition, even the minimal basis functions have to be disabled in the `species_defaults` for this atom type in the `control.in` (see next section). To distinguish between real and empty atoms, the species name has to be changed.

The following example for Zn_2^+ shows the 3 different `geometry.in`-files.

Fragment 1, Zn^+

```
atom 0.00 0.00 0.00 Zn
initial_charge 1
empty 0.00 0.00 5.00 Zn_empty
```

Fragment 2, Zn

```
empty 0.00 0.00 0.00 Zn_empty
#initial_charge 1 #no charge for emtpy atoms
atom 0.00 0.00 5.00 Zn
```

Complete System, Zn_2^+

```
atom 0.00 0.00 0.00 Zn
initial_charge 1
atom 0.00 0.00 5.00 Zn
```

control.in:

For every atom type of the embedded system, the species_data part has to be included and the species name must be changed to *species_empty*. Disable the minimal basis with the keyword `species` `include_min_basis` set to `.false.` and comment **all** tier basis function lines, as showed in the following example for Zn.

```
[...]
  species        Zn_empty
#     global species definitions
    nucleus             30
    mass                65.409
#
    l_hartree           4
#
    cut_pot             3.5          1.5  1.0
    basis_dep_cutoff    1e-4
#
```

```
    include_min_basis .false.

[...]

#  "First tier" - improvements: -270.82 meV to -12.81 meV
     #hydro 2 p 1.7
     #hydro 3 s 2.9
     #hydro 4 p 5.4
     #hydro 4 f 7.8
     #hydro 3 d 4.5
##  "Second tier" - improvements: -3.35 meV to -0.82 meV
     #hydro 5 g 10.8
     #hydro 2 p 2.4
     #hydro 3 s 6.2
     #hydro 3 d 3
```

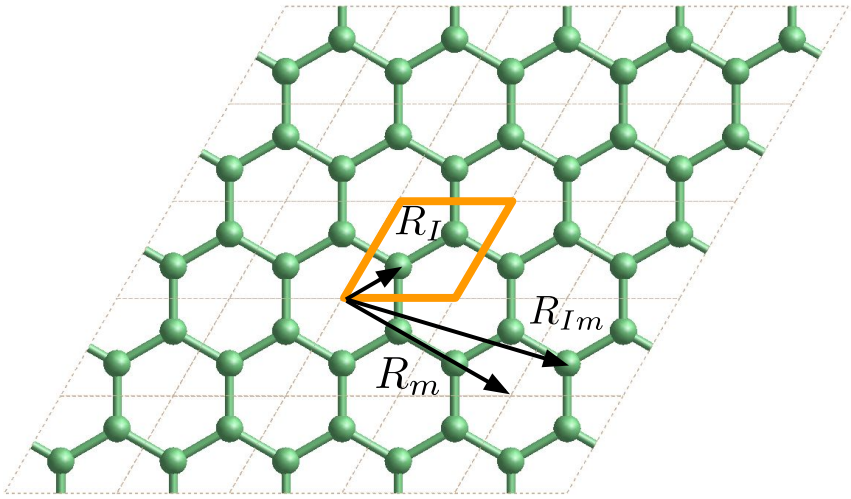


Figure 3.19: . Illustration of atomic coordinates in the unit cell ($\mathbf{R}_I$), lattice vector ($\mathbf{R}_m$) and atomic coordinates in super cell ($\mathbf{R}_{Im} = \mathbf{R}_I + \mathbf{R}_m$). Image credits: Honghui Shang.

3.40 Symmetry

This functionality is considered "under development". Please report any errors to the issue tracker at https://aims-git.rz-berlin.mpg.de/. No post-processing (DOS, band-structure) implemented, yet. Only realspace EXX.

FHI-aims can make use of symmetry to reduce the number of k-points and thus the number of calls to the eigensolver. The functionality is based on the external library *spglib* written by Atsushi Togo (**https://atztogo.github.io/spglib/**). For instructions how-to compile *FHI-aims* with *spglib* see Appendix E.2.

In real space, *FHI-aims* uses the Bloch functions

$$\chi_{I(\mu),\mathbf{k}}(\mathbf{r}) = \sum_m \varphi_{I(\mu),m}(\mathbf{r}) \exp(-i\mathbf{k}\mathbf{R}_m) \ , \tag{3.184}$$

where

$$\varphi_{I(\mu),m}(\mathbf{r}) = \varphi_{I(\mu),m}(\mathbf{r} - \mathbf{R}_{I(\mu)} - \mathbf{R}_m) \tag{3.185}$$

is the local basis function (atomic orbital) with the quantum numbers $\mu = (n, l, m)$ associated with atom $I(\mu)$ in the m^{th} periodic replica of the unit cell (see Fig. 3.19). The general space group symmetry operator has the form:

$$\hat{S} = \{V|\mathbf{R} + \mathbf{f}\} \tag{3.186}$$

V are rotations, $\mathbf{f}$ (fractional) translation vectors and $\mathbf{R}$ direct lattice translation vectors. Under such symmetry operations $\{V|\mathbf{f}\}$, the local basis function in equation 3.185 transform according to:

$$\{V|\mathbf{f}\}\varphi_{I(\mu),m}(\mathbf{r}) = \tag{3.187}$$

$$\varphi_{I(\mu),m}(V^{-1}\mathbf{r} - V^{-1}(\mathbf{R}_{I(\mu)} + \mathbf{R}_m + \mathbf{f})) \ . \tag{3.188}$$

We make use of the fact that Atom I of the first unit cell is transformed by application of $V^-1|\mathbf{f}$ into atom J in the jth unit cell. Also, $\mathbf{O}_I^V$ is the vector translating $\mathbf{R}_{J(\mu)}$ back to the first unit cell ($V^{-1}(\mathbf{R}_{I(\mu)} - \mathbf{f}) \rightarrow \mathbf{R}_{J,c}; \mathbf{O}_I^V = \mathbf{R}_{J,0} - \mathbf{R}_{J,c}$):

$$\begin{aligned} \mathbf{R}_{I(\mu)} + \mathbf{R}_m &= V(\mathbf{R}_{I(\mu)}^V + \mathbf{R}_m^V) + \mathbf{f} && (3.189) \\ \mathbf{R}_{I(\mu)}^V &= V^{-1}(\mathbf{R}_{I(\mu)} - \mathbf{f}) + \mathbf{O}_I^V = \mathbf{R}_{J(\mu')} && (3.190) \\ \mathbf{R}_m^V &= V^{-1}\mathbf{R}_m - \mathbf{O}_I^V = \mathbf{R}_j && (3.191) \end{aligned}$$

Using this, we finally get:

$$\{V|\mathbf{f}\}\varphi_{I(\mu),m}(\mathbf{r}) = \sum_{m'} \hat{T}(V,l,m,m')\varphi_{J(\mu'),j}(\mathbf{r} - \mathbf{R}_{J(\mu')} - \mathbf{R}_j) \;, \tag{3.192}$$

in which $\hat{T}(V,l,m,m')$ is the transformation matrix of the spherical harmonics Y_{lm}. Along the same line, the Bloch states in Eq. (3.184) fulfill

$$\mathbf{k}\mathbf{R}_m = (V^{-1}\mathbf{k})(V^{-1}\mathbf{R}_m + \mathbf{O}_I^V - \mathbf{O}_I^V) = \mathbf{k}^V\mathbf{R}_m^V + \mathbf{k}^V\mathbf{O}_I^V \tag{3.193}$$

so that we get

$$\{V|\mathbf{f}\}\chi_{I(\mu),\mathbf{k}}(\mathbf{r}) = \exp\left(-i\mathbf{k}^V\mathbf{O}_I^V\right)\sum_{m'} \hat{T}(V,l,m,m')\chi_{J(\mu'),\mathbf{k}^V}(\mathbf{r}) \tag{3.194}$$

Please note that the KS eigen-coefficients transform exactly like the basis functions:

$$c_{i,I(\mu),\mathbf{k}} = \exp(-i\mathbf{k}^V\mathbf{O}_I^V)\sum_{m'} \hat{T}(V,l,m,m')c_{i,J(\mu'),\mathbf{k}^V} \tag{3.195}$$

$$c_{i,J(\mu'),\mathbf{k}^V} = \exp(i\mathbf{k}^V\mathbf{O}_I^V)\sum_{m} \hat{T}^+(V,l,m,m')c_{i,I(\mu),\mathbf{k}} \tag{3.196}$$

The symmetry analysis, k-point reduction and reconstruction of the density matrix is done as follows:

1. We use *spglib* (http://atztogo.github.io/spglib/) to

   - determine the space group of the system
   - determine translation vectors ($\mathbf{f}$) and rotation matrices (V) for this space group

   See Appendix E.2 how-to include *spglib* during the compilation of *FHI-aims*. Additionally, we implemented routines to

   - tabulate the rotation matrices (V) required for the calculation of $\hat{T}(V,l,m,m')$.
   - and calculate the distance $\mathbf{O}$ between equivalent atoms due to the symmetry operation $\{V|\mathbf{f}\}$ to determine the phase factor in Eq. 3.196.
   - determine the equivalent $\mathbf{k}$-points
   - construct the map between symmetry-equivalent $\mathbf{k}$-points

2. The Euler angles are calculated from the rotation matrices $V(\alpha,\beta,\gamma)$.

   $$V(\alpha,\beta,\gamma) = \begin{pmatrix} \cos\gamma\cos\beta\cos\alpha - \sin\gamma\sin\alpha & \cos\gamma\cos\beta\sin\alpha + \sin\gamma\cos\alpha & -\cos\gamma\sin\beta \\ -\sin\gamma\cos\beta\cos\alpha - \cos\gamma\sin\alpha & -\sin\gamma\cos\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta \\ \sin\beta\cos\alpha & \sin\beta\sin\alpha & \cos\beta \end{pmatrix},$$

   This corresponds to the "y-convention":

   1. The x_1'-, x_2'-, x_3'-axes are rotated anticlockwise through an angle α about the x_3 axis
   2. The x_1''-, x_2''-, x_3''-axes are rotated anticlockwise through an angle β about the x_2' axis
   3. The x_1'''-, x_2'''-, x_3'''-axes are rotated anticlockwise through an angle γ about the x_3'' axis

3. The rotation matrices $\hat{T}(V,l,m,m') = \hat{T}(V(\alpha,\beta,\gamma),l,m,m') = \hat{T}^l_{m,m'}(\alpha,\beta,\gamma)$ for the **real** spherical harmonics ($\hat{T}(V,l,m,m')$) are calculated.
   The rotation matrix in the basis of the **real** spherical harmonics is calculated from the rotation matrix in the basis of the **complex** spherical harmonics, the Wigner D-Matrix $D^l_{mm'}(\alpha,\beta,\gamma)$, and a transformation matrix $\hat{C}^l_{m,m'}$ (see *M. A. Blanco, M. Florez, M. Bermejo, J. of Mol. Strucure, 419 (1997) 19-27*, [27]):

   $$\hat{T}^l_{m,m'}(\alpha,\beta,\gamma) = \hat{C}^l_{m,m'}{}^* \hat{D}^l_{mm'}(\alpha,\beta,\gamma)\hat{C}^l_{m,m'}{}^T$$

   - The Wigner D-Matrix (rotation matrix in the basis of the **complex** spherical harmonics) is defined by the formula:

     $$D^l_{mm'}(\alpha,\beta,\gamma) = \sum_i \frac{(-1)^i\sqrt{(l+m)!(l-m)!(l+m')!(l-m')!}}{(l-m'-i)!(l+m-i)!i!(i+m'-m)!} \times \left(\cos\frac{\beta}{2}\right)^{2l+m-m'-2i}\left(\sin\frac{\beta}{2}\right)^{2i+m'-m} e^{-i(m\alpha+m'\gamma)},$$

     (from "Bradley and Cracknell, The mathematical theory of symmetry in solids : representation theory for point groups and space groups, Clarendon Pr., 1972, p.53", [33]) Thereby, improper rotations (combination of rotation and inversion) are made proper $R \to -R$ and $D^l_{mm'} \to (-1)^l D^l_{mm'}$. In practice the calculation of $D^l_{mm'}(\alpha,\beta,\gamma)$ is done with a recursive algorithm, that does not require the calculation of factorials. (*M. A. Blanco, M. Florez, M. Bermejo, J. of Mol. Strucure, 419 (1997) 19-27*, [27])
   - $\hat{C}^l_{m,m'}$ is constructed by these 6 rules:
     1. $\hat{C}^l_{m,m'} = 0$ if $|m| \neq |m'|$
     2. $\hat{C}^l_{0,0} = 1$
     3. $\hat{C}^l_{m,m} = (-1)^m/\sqrt{2}$
     4. $\hat{C}^l_{m,-m} = 1/\sqrt{2}$

5. $\hat{C}^l_{-m,m} = -\mathrm{i}(-1)^m/\sqrt{2}$
6. $\hat{C}^l_{-m,-m} = \mathrm{i}/\sqrt{2}$

- Last but not least, we have to take care of the sign convention for the real spherical harmonics as implemented in FHI-aims – figuring this out took us some time. In practice, this is taken care of by an additional matrix multiplication (with $T^l_{m,m'}$) yielding the correct signs in $\hat{C}^l_{m,m'}$.

$$\hat{C}^l_{m,m'} = T^l_{m,m'} \times \hat{C}^l_{m,m'}$$

4. Eventually, this matrices are used to transform the KS eigen-coefficients following Eq. (3.196). Furthermore, the rotation of the KS-eigenvectors can be made more efficient by directly transforming the density matrix $n(\mathbf{k}, n, m)$ at each k-point with the help of two matrix operations:

$$\begin{aligned} n(\mathbf{k}, n, m) = \sum_{i,j}^{occ} c^*_{i,n,\mathbf{k}} c_{j,m,\mathbf{k}} &= \sum_{i,j}^{occ} \sum_{n'} exp(i\mathbf{k}^V \mathbf{O}^V_{n'}) T^*_{n,n'} c^*_{i,n',\mathbf{k}^V} \\ &\times \sum_{m'} exp(-i\mathbf{k}^V \mathbf{O}^V_{m'}) T_{m,m'} c_{j,m',\mathbf{k}^V} \\ &= \hat{T}^* n(\mathbf{k}^V, n', m') \hat{T}^T \end{aligned} \tag{3.197}$$

The phase factor $exp(i\mathbf{k}^V \mathbf{O}^V_{n'})$ can be included in the transformation matrix during the pre-processing. This increases the matrix size (and required memory) by a factor of the size of the number of k-points to reconstruct at each computing task. Furthermore the density matrix only has to be reconstructed for k-points reduced by proper rotations. The corresponding improper (inversion symmetry) rotations are accounted for by the integration weights.

Tags for general section of `control.in`:

Tag: `symmetry_reduced_k_grid_spg` (control.in)

Usage: `symmetry_reduced_k_grid_spg .true./.false.`

Purpose: Only use the irreducible set of k-points during the calculation. Default: `.false.`

Tag: `reconstruct_proper_only` (control.in)

Usage: `reconstruct_proper_only .true./.false.`

Purpose: Only reconstruct the density matrix for proper rotations. Improper rotations (Inversion symmetry) are accounted for by the integration weights. Default: `.true.`

Tag: `use_spg_full_Delta` (control.in)

Usage: `use_spg_full_Delta .true./.false.`

Purpose: Include phase factors in the reconstruction matrix for the density matrix during pre-processing. Set to `.false.` if memory is an issue. Default: `.true.`

Tag: `use_spg_mv_mm` (control.in)

Usage: `use_spg_mv_mm .true./.false.`

Purpose: Reconstruct the density by rotating the eigenvector and setting up the density matrix in the standard way (matrix-vector and matrix-matrix operation instead of the matrix-matrix operations in Eq. 3.197). Default: `.false.`

Tag: `use_symmetric_forces` (control.in)

Usage: `use_symmetric_forces .true./.false.`

Purpose: Symmetrize the forces and generalized forces on the lattice, i. e., the stress, for geometry relaxation, e. g., to preserve crystal symmetry but without fully reducing the k-point set. If full symmetry is used for k-point reduction, forces are symmetrized. The forces are symmetrized by averaging over all symmetry operations. Default: `.false.`

There keywords here are RLSY based symmetry related. It is a work by Yi Yao, Olle Hellman, and Volker Blum to use spacegroup symmetry to accelerate the DFT calculation. Please report bugs by opening issue in aims git repository.

Tag: `rlsy_symmetry` (control.in)

Usage: `rlsy_symmetry all`

Purpose: grid based symmetry reduction calculation to accelerate DFT calculations. Default: `None`

Tag: `rlsy_symmetry_refine_structure` (control.in)

Usage: `rlsy_symmetry_refine_structure .true./.false.`

Purpose: refine the structure and lattice vectors to idealized position based on the spacegroup symmetry. Default: `.false.`

Tag: `hartree_d_matrix_method` (control.in)

Usage: `hartree_d_matrix_method pseudo_inverse/submatrix`

Purpose: The original d matrix inversion method for Hartree potential calculation described in Delley's paper is the submatrix method. It would introduce slightly asymmetricity in the calculation especially for stress tensor. Swithing it to the pseudo inverse method can reach a fully symmetry result with some additional calculation. Default: `submatrix`

3.41 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`:

Tag: `verbatim_writeout` (geometry.in)

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: `cube_default_size_safeguard` (control.in)

Usage: `cube_default_size_safeguard number`

Purpose: Sets the maximum size that a cube output file is allowed to have if its dimensions are based on internal defaults.

`number` is an integer number. Default: `number`=$5{\cdot}10^7$

When the `output cube` functionality is requested in order to print a three-dimensional array on an even-spaced grid, FHI-aims can use internal defaults to determine the dimensions of this cube file. Each cube file is based on an even-spaced grid with n grid points, where $n = n_1 \times n_2 \times n_3$ and n_i is the number of grid segments used along each of the grid directions i=1, 2, 3. The `output cube` edge subtag can be used to specify the grid spacing. However, if that subtag is not set, a default value of 0.1 Å is used for discretization of the three cube edges. For large structures, this could lead to very large cube file sizes. As a safeguard, FHI-aims stops when the number of points in a cube, n, exceeds the number set by `cube_default_size_safeguard`. Its default value, $5{\cdot}10^7$, corresponds to a cube file size of about 800 MB (i.e., 16 bytes of information are stored for each point of a cube grid). If the edges of any cube files are set explicitly using the `output cube` edge subtag, the criterion set by `cube_default_size_safeguard` is not checked and the code does noty stop.

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.

Restriction: Works only for periodic systems. Does not work when keyword `use_local_index` is set.

Only useful in conjunction with the keywords `output dos`, `output dos_tetrahedron` or `output postscf_eigenvalues`, 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 $10\times10\times10$ k-point grid, but subsequently, a density of states is computed with an $80\times80\times80$ k-point grid.

Note that this additional calculation is done using serial solutions of the eigenvalue problems for individual k-points on individual CPU cores. This always works but will create memory problems as the system size increases, simply because local copies of all matrices are kept on single CPUs. For large systems, our usual, more sophisticated parallelization strategies have not yet been copied over to this routine.

Tag: `elsi_output` (control.in)

Usage: `elsi_output verbosity`

Purpose: Controls the output level of ELSI.

`verbosity` is a keyword (string). Default: `detail`.

Available options for `verbosity` are:

- `none` : No output from ELSI. This is the default if the overall output level of FHI-aims is `MD_light`.
- `light` : Enables output from the ELSI interface, but no output from the solvers.
- `detail` : Enables output from the ELSI interface as well as the solvers. When using libOMM or PEXSI, additional output will be written to an separate log file.

- `debug` : Enables the same output as does the `detail` option, with additional memory usage information. Creates large output files and thus should not be chosen in production runs.
- `json` : Enables the output of the runtime parameters used in ELSI in a separate JSON file, powered by the FortJSON library. May be used on top of the above options.

Tag: `elsi_output_matrix` (control.in)

Usage: `elsi_output_matrix matrix`

Purpose: Outputs the k-space Hamiltonian, overlap, density matrices, and KS eigenvectors in the ELSI format.

`matrix` is a string, specifying the desired matrix to output.

Available options for `matrix` are:

- `hamiltonian` : Outputs the SCF converged Hamiltonian matrix.
- `overlap` : Outputs the overlap matrix.
- `density_matrix` : Outputs the SCF converged density matrix.
- `eigenvectors` : Outputs the SCF converged KS eigenvectors.

Note that this keyword outputs matrices in the k-space instead of the real space. Therefore, the number of matrix files would be equal to the number of k-points, multiplied by the number of spin channels. The output files are in the ELSI CSC binary format (see the documentation of ELSI). The Python scripts in the "utilities/elsi_matrix" directory may be used to post-process an ELSI matrix file, e.g., converting it to a human-readable format.

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 at the end of the first SCF step and at the end of the last SCF step where the chemical potential is included too. For ease of evaluation, the self-consistent work function will be repeated again at the end of the output file in the summary block.

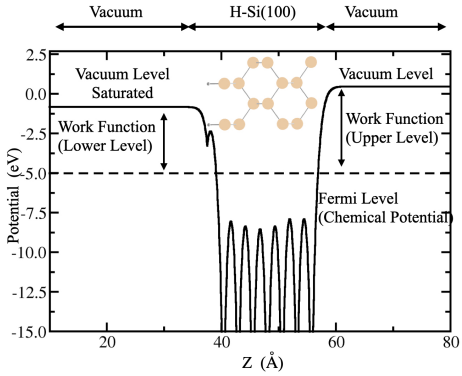


Figure 3.20: The electrostatic potential for the Si100 surface, saturated with hydrogen on one side. The Fermi level is at -4.94 eV, the vaccum of the unsaturated (upper) and saturated (lower) surface is evaluated as 0.46 eV and -0.82 eV respectively.

Note that for *non-symmetric* slabs, the upper and lower work function should generally be different. This behavior is shown in Fig 3.20, the work function is defined as the difference between the Fermi surface and vacuum level which are different on the saturated and unsaturated surfaces. Accordingly, the work function exhibits the same asymmetry. In practice, this behavior is correctly reproduced only if the `use_dipole_correction` is additionally specified.

If the `include_spin_orbit` is requested, the work function output, both neglecting and including the post-SCF SOC corrections, will be printed directly after the output of the *SOC calculation*.

If the work function output is requested, keyword `compensate_multipole_errors` is now automatically switched on by default. This will change total energies slightly compared to the uncompensating case, and – we believe – even for the better. It will certainly lead to a better description of the long-range Hartree potential.

However, it must be possible to find a vacuum plane z, where the surface dipole is compensated, that is further than 6 Å away from the nearest atom. Otherwise, the calculation will stop and alert the user.

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. One 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 subsubsection below.

Tag: output_boys_centers (control.in)

Usage: output_boys_centers

Purpose: Calculates and outputs the maximally localized Boys centers (equivalent of Wannier centers for the isolated molecule case).

The maximization procedure follows JCP 135, 134107 (2011). Currently only the cartesian position of the centers is outputted in xyz format in geometry_boys.xyz. The transformation matrix is also calculated, but only applied to the orbitals when using the keyword apply_boys .

Tag: output_cube_nth_iteration (control.in)

Usage: output_cube_nth_iteration n

Purpose: Writes all cube files specified in control.in every nth SCF iteration.

n is an integer greater than or equal to 1. Default: N/A (cube files will be output after the SCF cycle has converged.)

By default, cube files are written once, after the SCF cycle has converged. With this keyword,all cube files specified in control.in will be written each n^{th} iteration, where n is an integer greater than zero. This keyword should be helpful to analyse what is going on during subsequent SCF cycles. However, the output of cubes is usually quite slow, so choosing this option will slow down the calculation a lot.

This keyword does not support spin-orbit coupling, as spin-orbit coupling is applied after the SCF cycle has converged.

This keyword is only applicable when the output cube keyword(s) are being used; please see the manual entry for output cube for more information.

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 dymnamics 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: `overwrite_existing_cube_files` (control.in)

Usage: `overwrite_existing_cube_files flag`

Purpose: Allows overwriting of pre-existing cube files with new cube files of the same name, instead of preserving the pre-existing cube files by appending numbers to the end of the new file names.

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

If set to .false., FHI-aims will check during the output of cube files whether a file with the selected file name already exists. If such a file is found, the file name will be changed by adding a number (1,2,3...) to the end of the file name. This is very useful when relying on default names or when plotting cubes during the SCF cycle.

If set to .true. FHI-aims will not check during output whether a file with the selected name already exists. If it does exist, it will be simply overwritten!

This keyword is only applicable when the `output cube` keyword(s) are being used; please see the manual entry for `output cube` for more information.

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 output `types` available through the `output` keyword:

`output` **sub-tag:** `acks2_parameters` (control.in)

Usage: `output acks2_parameters`

Purpose: Evaluate the KS-DFT electronic structure based Cartesian Gaussian basis set parameters (xc-contributions to hardness and non-interacting linear respone kernel) of the ACKS2 model [88] for the given atomic structure and write them to file. Definition of the ACKS2 density and KS-potential basis set options are to be provided in an additional file called 'acks2.in'. An example file is given below, add more lines for each individual atom of the geomtry.in file and update the number of radial basis functions accordingly (remove all comments started by #). The read format specifier for the Gaussian width parameters is '(A4, F12.6)', i.e. four horiztonal white space before angular type definition (s, p, d, etc.) and F12.6 floating point representation for Gaussian width parameter.

Illustrative file content 'acks2.in' for H_2 structure:

finite_diff_epsilon 01.0E-07 # central difference scheme size
threshold_radial 01.0E-07 # radial threshold for Gaussian function evaluation

density_basis 00004 # keyword plus total number of radial functions
02 s 0000.264085 p 0000.691120 # 1st atom, 2 radial functions, s-type and p-type
02 s 0000.264085 p 0000.691120 # 2nd atom, 2 radial functions, s-type and p-type

KS_potential_basis 00004 # keyword plus total number of radial functions
02 s 0000.228849 p 0000.836703 # 1st atom, 2 radial functions, s-type and p-type
02 s 0000.228849 p 0000.836703 # 2nd atom, 2 radial functions, s-type and p-type

Restrictions: This functionality is available only for *non-periodic* systems and *non-spin-polarized* systems. The exchange-correlation functional implementation of ACKS2 has only been targeted and tested for GGA (and LDA) versions.

`output` **sub-tag:** `aitranss` (control.in)

Usage: `output aitranss`

Purpose: Writes Kohn-Sham eigenvectors c_{il} and energies ε_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 parallel` 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.

Note: You should no longer need the `output species_proj_dos` , `output atom_proj_dos` or `output dos` at all. Instead, the same objects can be obtained with MUCH better integration accuracy using the alternative keywords `output species_proj_dos_tetrahedron` , `output atom_proj_dos_tetrahedron` or `output dos_tetrahedron` . Please see there for the syntax. The description of the older keywords is kept for now.

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 $\boldsymbol{G}$=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.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled DOS will have the default filename(s) and the file(s) containing the scalar-relativistic (i.e. no SOC) DOS will have an additional suffix ".no_soc". Note that if you requested `spin collinear` in the `control.in` file, there will be only *one* file per atom (and equivalently for all `_raw.dat` files) containing the projected DOS of all spin-coupled states:

- `atom_proj_dos_`*`number`*`.dat`

The reason is that the separate spin channels of scalar relativity no longer exist after the SOC operator is applied – the states now form a single set. However, there are two DOS files for the output without spin-orbit coupling; one for each spin channel:

- `atom_proj_dos_spin_up`*`number`*`.dat.no_soc`
- `atom_proj_dos_spin_dn`*`number`*`.dat.no_soc`.

`output` **sub-tag:** `atom_proj_dos_tetrahedron` (control.in)

Usage: `output atom_proj_dos_tetrahedron Estart Eend n_points`

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.

This is the keyword that should be used to obtain atom-resolved densities of states with high integration resolution.

This option is based on a Mulliken Analysis and shares its syntax with `output dos_tetrahedron` and `output species_proj_dos_tetrahedron` . See also section 4.4 for more details.

There are two types of output files for each atom:

- atom_proj_dos_*number*_tetrahedron_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 $\boldsymbol{G}$=0 component of the long-range Hartree potential (periodic systems).
- atom_projected_dos_*number*_tetrahedron.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.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled DOS will have the default filename(s) and the file(s) containing the scalar-relativistic (i.e. no SOC) DOS will have an additional suffix ".no_soc". Note that if you requested `spin collinear` in the control.in file, there will be only *one* file per atom (and equivalently for all tetrahedron_raw.dat files) containing the projected DOS of all spin-coupled states:

- atom_proj_dos_tetrahedron_*number*.dat

The reason is that the separate spin channels of scalar relativity no longer exist after the SOC operator is applied – the states now form a single set. However, there are two DOS files for the output without spin-orbit coupling; one for each spin channel:

- atom_proj_dos_tetrahedron_spin_up*number*.dat.no_soc
- atom_proj_dos_tetrahedron_spin_dn*number*.dat.no_soc.

`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 <kstart1,kstart2,kstart3> to <kend1,kend2,kend3> 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.

The band structure output files are named `bandXYYY.out`, where the letters X and YYY are replaced with numbers in the actual output file names:

- The letter X encode the spin channel. In a non-spinpolarized or in a spin-orbit coupled calculation, X will always be 1. In a spin-polarized calculation, X=1 indicates the first spin channel, X=2 indicates the second spin channel.
- The letters YYY are the consecutive numbers of the bands (starting from 001) requested in the `control.in` file, that is, the bands given in the order of `output` lines in `control.in`.

The files `bandXYYY.out` have the format

`ipoint k1 k2 k3 occ1 E1 occ2 E2 ... occN EN`,

i.e. they specify not only the bands for each k-point but also the occupation number for this particular point.

A safe starting value for `n_points` when performing semi-local calculations is 21. We have found that this generally samples the fine features of the k-path reasonably well, even for small unit cells with correspondingly large Brillouin zones. For hybrid-functional calculations, due to the computational expense one should consider using a smaller value. We also note that `n_points` includes the end-points, i.e. `n_points`=21 will give 20 intervals for a given branch. For comparison with results calculated by other DFT codes, it's recommended to use values of form 1, 6, 11, 16, 21, ... to ensure that the reciprocal coordinates are nice, simple fractions.

Note that a fully occupied band has the occupation number 2.0 in a non-spinpolarized calculation. In a spin-polarized or spin-orbit-coupled calculation, the maximum occupation number is 1.0.

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 calculations, the eigenvectors, overlap matrices, and hamiltonian matrices at each k-point requested by `output band` can be written out using the `output eigenvectors`, `output overlap_matrix`, and `output hamiltonian_matrix` keywords, respectively.

For periodic band structure output, the `exx_band_structure_version` keyword must be set – see the respective Section 3.24 for a brief explanation of the background.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled band structures will have the default filename(s)

and the file(s) containing the scalar-relativistic (i.e. no SOC) band structures will have an additional suffix ".no_soc". Note that if you requested `spin collinear` in the `control.in` file in addition, there will be only *one* file per band segment containing all spin-coupled states (output with spin-orbit coupling):

- `band1YYY.dat`

The reason is that the separate spin channels of scalar relativity no longer exist after the SOC operator is applied – the states now form a single set. However, there are two band segment files for the output without spin-orbit coupling; one for each spin channel:

- `band1YYY.dat.no_soc`
- `band2YYY.dat.no_soc`

---

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 <kstart1,kstart2,kstart3> to <kend1,kend2,kend3> 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 `output band` , 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 `output band` functionality is experimental or, for some reason, not available.

---

output sub-tag: band_mulliken (control.in)

Usage: `output band_mulliken kstart1 kstart2 kstart3 kend1 kend2 kend3 n_points name_start name_end`

Purpose: Plots a band along the line from <kstart1,kstart2,kstart3> to <kend1,kend2,kend3> at `n_points` equally spaced points. The k-vectors are written in relative coordinates of the reciprocal basis vectors.

This keyword allows to calculate the mulliken charge analysis at all K points along the band K path. The file name is named as bandmlk1001.out, bandmlk1002.out, ..., etc. In the output file, the mulliken charge data is written first by K point, then by eigenstates. In each eigenstate, the mulliken charge on all atoms is written line by line. In each

line, the following information is written: eigenstate ID, eigenvalue, occupation number, atom ID, spin ID, total mulliken charge, mulliken charge for l = 0, 1, 2,.. etc. For each state at a given point, the total mulliken charge for all atoms should sum up to 1 or 2, for non-SOC and SOC, respectively.

There is a keyword called `band_mulliken_orbit_num` specifying how many orbitals (states) to be written out. If the number following the keyword `band_mulliken_orbit_num` is I, then the orbitals in the range of HOMO - I + 1 and LUMO + I would be written out in the output file. The default value of `band_mulliken_orbit_num` is 50 for non-SOC and 100 for SOC.

To plot the band structures with Mulliken decomposition, two python script in the utilities directory in FHI-aims distribution can be used, i.e., band_mlk.py and band_mlk_soc.py for non-SOC and SOC, respectively. Running of these scripts is instructed in the first lines of these python files.

`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:

- `A`i`_`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.
- `A`i`_`j`_`nl`_base_kin.dat` : Corresponding kinetic energy expression $[\epsilon - v(r)] \cdot u(r)$
- `C`i`_`j`_`nl`_base.dat` : `confined` free-atom like radial function $u(r)$ number j for `species` number i, radial and angular quantum numbers nl.
- `C`i`_`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 `A`i`_`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.
- Si_j_nl_base.dat : Slater-type orbital radial function $u(r)$ for species number i, radial function number j, radial and angular quantum numbers nl.
- Si_j_nl_base_kin.dat : Corresponding kinetic energy expression $[\epsilon - v(r)] \cdot u(r)$

---

output sub-tag: batch_statistics (control.in)

Usage: output batch_statistics

Purpose: Write out statistics for each batch of points used in the evaluation of real-space quantities, organized by associated MPI task (one file per MPI task)

This keyword outputs information about the batch distribution used by FHI-aims to evaluate real-space quantities (real-space Hamiltonian, charge density update, etc.) Every MPI task creates a batch_statistics_task_###.dat file, where ### is the MPI task's rank, and statistics for each batch are output sequentially to file. Statistics output for each batch include:

- Number of points in the batch
- Minimum and maximum number of radial basis functions evaluated on batch
- Maximum number of basis functions evaluated on batch

- Minmum and maximum number of atoms whose basis functions are evaluated on the batch
- Minimum and maximum values for the integration weights for points in batch

Note: As of this writing, the output files will be rewritten with every SCF restart, including SCF reinitialization, geometry relaxation steps, and MD steps.

---

output sub-tag: cc4s (control.in)

Usage: output cc4s

Purpose: Activates the calculation and output of quantities relevant for a subsequent Coupled cluster calculation via the CC4S code (https://manuals.cc4s.org/user-manual/index.html). In order to use this option, FHI-aims must have been compiled with the CC-aims interface (https://gitlab.com/moerman1/fhi-cc4s).

This keyword can be specified in combination with a SCF-calculation and will calculate and/or write to file quantities CC4S requires after the SCF has been completed.

If the xc -functional used is Hartree-Fock the output files generated by CC-aims can be used by CC4S to perform a canonical Coupled cluster calculation. In principal, one can also specify different xc -functionals and the quantities for a non-canonical Coupled cluster calculation will be calculated correctly. However, at this point in time, the non-canonical Coupled cluster algorithms are not yet implemented.

Note, that the CC-aims interface relies on ScaLAPACK, so that KS_method parallel must be used.

A number of additional keywords have been added to FHI-aims to give the user more control over the calculation of the CC4S-quantities. These can be found in Section CC4S.

---

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/

It is accepted by multiple visualization tools.

Visualization tools which may be used to plot cube file and are available for all major operating systems are Avogadro (https://avogadro.cc), jmol (http://www.jmol.org), and VMD (http://www.ks.uiuc.edu/Research/vmd/). For periodic systems, excellent success has been reported by multiple users using Vesta (https://jp-minerals.org/vesta/en/). Please see the documentation of those programs for more information on plotting the resulting cube files.

By default, the cube files will be output once, after the SCF cycle has converged, and FHI-aims will avoid overwriting pre-existing cube files it finds by appending numbers to the end of new file names. To output the cube files at regular intervals during the SCF cycle, use the output_cube_nth_iteration keyword, and to overwrite pre-existing cube files with new files, use the overwrite_existing_cube_files keyword.

This keyword supports spin-orbit coupling, but only when the type is either eigenstate_density or eigenstate. The large-scale use_local_index and load_balancing keywords are only supported when the type is either eigenstate_density or eigenstate.

Before we move on to the supported options for type, as well as other keywords related to cube output, there is an important note about specifying the dimensions of the cube that all users should know.

If no cube grid spacings are specified using the output cube edge tag, FHI-aims will use its own internal default for this grid spacing. In some cases, such as separated molecules or surfaces with large amounts of vacuum, the cube dimensions that FHI-aims would silently default to may not be ideal and may result in excessively large files.

In order to prevent uncontrolled damage (such as, filling up a file system or quota to beyond any reasonable limits) the code will therefore stop if a default number of cube grid points would be written to a single file in excess of limiting value defined by keyword cube_default_size_safeguard . This limit is configurable (see the description of that keyword).

If any output cube edge tag is specified in control.in, the limit given by cube_default_size_safeguard does not apply.

Also, many viewers do not implement non-rectangular cube edges, which would result from non-rectangular unit cells by default.

In short: Users are *always* strongly encouraged to specify cube geometries directly.

A list of all keywords related to cube plotting is given below. After this list of keywords, we have provided an example set of lines for plotting the total density of a molecule as well as the densities for individual eigenstates. This example should be adaptable for other types of cube files. Units for densities and eigenstates are $Å^{-3}$ and $Å^{-3/2}$, respectively. The unit for the long range and hartree potential is Hartree [Ha]. However, for electronic density response the unit is $VÅ^{-2}$.

- 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_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 contribution from the imaginary part of the wave function, and thus the output of this `type` is more physically relevant than the `eigenstate type`, which outputs only the real part of the wave function. By default, the first spin channel and the first k-point is printed out, see also `cube spinstate`. In the spin-orbit coupled case, electron density contributed from both spin channels and the first k-point is printed out.

3. `eigenstate` n : Writes the *real part* of the wave function of the n-th eigenstate to a file. n must be an integer number. For non-periodic non-spin-orbit coupled calculations, the wave function has no imaginary part, so this `type` is sufficient. However, periodic and/or spin-orbit-coupled calculations produce wave functions with both a real part and an imaginary part. The corresponding imaginary part can be printed by requesting the cube type `eigenstate_imag` (see below) for the same state, in addition to the `eigenstate` type. Alternatively, one may `eigenstate_density type` for periodic calculations (and non-periodic calculations with spin-orbit coupling) instead. By default, the first spin channel and the first k-point is printed out, see also `cube spinstate` and `cube kpoint`.

4. `eigenstate_imag` n : If applicable (for example, in periodic or spin-orbit coupled calculations), writes the imaginary part of the of the n-th eigenstate to a file. n must be an integer number. Specifically for a spin-orbit coupled orbital, it is important to remember that such an orbital is a vector of two complex-valued functions:

$$\psi_{n,\mathrm{k}}(\boldsymbol{r}) = \begin{pmatrix} \psi^{(1)}_{n,\mathbf{k}}(\boldsymbol{r}) \\ \psi^{(2)}_{n,\mathbf{k}}(\boldsymbol{r}) \end{pmatrix} \tag{3.198}$$

The two functions $\psi^{(1)}_{n,\mathbf{k}}(\boldsymbol{r})$ and $\psi^{(2)}_{n,\mathbf{k}}(\boldsymbol{r})$ are complex-valued, i.e., they each have a real part and an imaginary part. Thus, a total of four scalar functions $\mathrm{Re}(\psi^{(1)}_{n,\mathbf{k}})$, $\mathrm{Im}(\psi^{(1)}_{n,\mathbf{k}})$, $\mathrm{Re}(\psi^{(2)}_{n,\mathbf{k}})$, $\mathrm{Im}(\psi^{(2)}_{n,\mathbf{k}})$ need to be plotted to get the full orbital (n,k). The two vector components (1) and (2) are sometimes loosely called "spin channels" although they are spin channels only in the non-relativistic limit; in the actual spin-orbit coupled case, the expectation value of the Pauli matrices would have to be calculated to get the spin direction. In any case, to get a full spin-orbit coupled orbital (say, n=1568 and the default k-point, usually – for unshifted k-point grids – Γ) as cube file output, the syntax to use in `control.in` is this:

```
output cube eigenstate 1568
cube spinstate 1

output cube eigenstate_imag 1568
```

```
cube spinstate 1

output cube eigenstate 1568
cube spinstate 2

output cube eigenstate_imag 1568
cube spinstate 2
```

5. `spin_density` : The spin density $n^{\uparrow}(\boldsymbol{r}) - n^{\downarrow}(\boldsymbol{r})$ is written to a file named `spin_density.cube`. Only available for `spin collinear`.
6. `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.
7. `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.
8. `total_density_integrable` : 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. The output value is the integration in the voxel instead of the value at the center of the voxel. The total density here is more suitable for functions like Bader analysis.
9. `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.
10. `hartree_potential`: The whole (i.e, short-range and long-range) electrostatic potential is recomputed on a cube grid and written out. Be careful with keyword `potential`. Please report errors.
11. `xc_potential`: The xc potential is recomputed on a cube grid and written out. **PBE only**, **spin unpolarized only**.
12. `potential`: Legacy. The whole (i.e, short-range and long-range) electrostatic potential is recomputed on a cube grid and written out. The keyword is considered broken/experimental. Use `hartree_potential`.
13. `delta_v`: Output $\delta v = v - v^{\mathrm{free}}$. This is especially tested for MPB solvation effects and is also an experimental feature especially for vacuum calculations.

14. `ion_dens`: Ionic charge density $n_{\mathrm{ion}}^{\mathrm{MPB}}$ as obtained from an MPB-DFT calculation. Still experimental.
15. `dielec_func`: Dielectric function $\varepsilon[n_{\mathrm{el}}]$ as used in the MPB-DFT calculation.
16. `elf`: Electron localization function. Different options are available for spin-polarized systems, see keyword `cube elf_type`. Currently under testing, please report any errors. The implementation is not yet compatible with spin-orbit calculations.
17. `first_order_density` n: Density response with respect to an applied homogeneous electric field is printed in a cube file ($\frac{\partial \rho}{\partial \epsilon_n}$). So that, n represents the direction of the field (1,2 or 3). a. For nonperiodic systems, `DFPT polarizability` should be specified in control.in before the cube file commands. b. For periodic system, `DFPT dielectric` should be specified in control.in before the cube file commands. An example outputting density response with respect to the three Cartesian direction in `control.in` is this:

```
output cube first_order_density 1
    cube origin       7.3408      7.6288     52.3975
    cube edge 100      0.1468      0.0000      0.0000
    cube edge 100      0.0000      0.1526      0.0000
    cube edge 300      0.0000      0.0000      0.1480
output cube first_order_density 2
    cube origin       7.3408      7.6288     52.3975
    cube edge 100      0.1468      0.0000      0.0000
    cube edge 100      0.0000      0.1526      0.0000
    cube edge 300      0.0000      0.0000      0.1480
output cube first_order_density 3
    cube origin       7.3408      7.6288     52.3975
    cube edge 100      0.1468      0.0000      0.0000
    cube edge 100      0.0000      0.1526      0.0000
    cube edge 300      0.0000      0.0000      0.1480
```

Hence, three cube file will be outputted for each direction.

- `cube spinstate` *spin*
  This keyword allows the user to choose whether to print spin-channel 1 or 2. The default value is 1. An example for how to use the `cube spinstate` keyword to print eigenstates in both spin channels is given below. This keyword is only useful for spin-polarized calculations (keyword `spin collinear`) and for spin-orbit coupled calculations. Otherwise, scalar-relativistic spin-non-polarized calculations (keyword `spin none`) have degenerate spin channels by definition. See the description of `eigenstate_imag` for an example of how to write all parts of a spin-orbit coupled orbital to a cube file.

- `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. Keyword `output k_point_list` may be used to print out the entire list of k-points used in the s.c.f. cycle. This will help identify which k-point is printed in a cube file. Default: 1

- `cube state` *spin k-point*
  Deprecated keyword to choose spin and k-point. The `cube spinstate` keyword should be used instead as given in the example below. If nothing else is specified, 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 paralellization 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` i j
  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: $i \cdot n^{\uparrow}(\boldsymbol{r}) - j \cdot n^{\downarrow}(\boldsymbol{r})$. 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 elf_type` i
  Specifies the type of electron localization function (ELF) to be calculated. The default $i = 0$ (and the only available option for spin-unpolarized calculations) is the Savin *et al.* formula [207]. For spin-polarized systems, $i = 1$ and $i = 2$ correspond to the original formulation by Becke and Edgecombe [18] for spin channels 1 and 2, respectively. If i is not 0, 1, or 2, and the system is spin-polarized, the Kohout-Savin variant of ELF [135] will be calculated.

- cube origin x y z
  Single line which specifies the origin , i.e., the center of the region to be plotted. Values are given in ÅIf 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 0.1 Å length, which span the whole molecule plus 14 Bohr beyond the outermost nucleii. For periodic calculations, the default edges are the same as the lattice vectors, again with 0.1 Å 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. In order to prevent uncontrolled damage (such as, filling up a file system or quota to beyond any reasonable limits) the code will therefore stop if a default number of cube grid points would be written to a single file in excess of limiting value defined by keyword cube_default_size_safeguard . This limit is configurable (see the description of that keyword).

  Users are *always* strongly encouraged to specify cube geometries themselves.

NOTE: A word of warning here: FHI-aims outputs the density and the wave function in units of $1/\text{Å}^3$ and $1/\text{Å}^{3/2}$, respectively, by default. In contrast, the length units of the cube voxels are in atomic units. The choice of units must be accounted for when doing any kind of postprocessing.

The keyword cube_content_unit allows one to switch the output units of the printed array.
cube_content_unit legacy is the default and gives the output in SI units, while
cube_content_unit bohr switches to output of densities and orbitals in atomic units. (Note that other codes' definition of cube file usually is in atomic units, which is why we do not provide a switch to Å).

As promised, an example set of lines for the control.in file showing how to plot the total density and densities for individual eigenstates of a hypothetical spin-polarized non-periodic system. This example will not exactly correspond to your system and should not be blindly copy-pasted. Nevertheless, it should give a good idea of how the keywords presented previously work together.

```
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_density 151
cube spinstate 1
output cube eigenstate_density 151
cube spinstate 2
output cube eigenstate_density 152
cube spinstate 1
output cube eigenstate_density 152
cube spinstate 2
output cube eigenstate_density 153
cube spinstate 1
output cube eigenstate_density 153
cube spinstate 2
output cube eigenstate_density 154
cube spinstate 1
output cube eigenstate_density 154
cube spinstate 2
```

The first line requests 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 densities correspond to the Kohn-Sham wave functions associated with eigenstates number 151-154. For each eigenstate, the additional `cube spinstate` *i* line requests first the spin-up channel (i=1), then spin-down (i=2).

NOTE that for periodic systems, the `cube spinstate` lines may have to be followed by `cube kpoint` lines to specify the k-point at which an eigenstate is printed. By default, only k-point number 1 is printed. For unshifted k grids, that is the Γ point.

Thus ends our brief treatise on cube plotting. We return you to your regularly scheduled programming.

`output` **sub-tag: `density`** (control.in)

Usage: `output density`

Purpose: Writes the electron density $n(\boldsymbol{r})$ at each integration grid point $\boldsymbol{r}$ 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:** `dgrid` (control.in)

Usage: `output dgrid`

Purpose: Dumps the Wave function of the final s.c.f. cycle on disk into the file *dgrid_aims.dat*. This files serves as an interface to the DGrid program.

This option serves as interface to the DGrid program (written by Miroslav Kohout), which is available free of charge at https://www2.cpfs.mpg.de/~kohout/dgrid.html. DGrid allows to employ various electronic structure and chemical bonding analysis alogorithms in real space, e.g., the QTAIM (Quantum Theory of Atoms in Molecules of R.F.W. Bader) or the ELI-D/ELF method. After installation of DGrid (Version ≥ 5.0) the command *dgrid dgrid_aims.dat* converts the interface wave function file *dgrid_aims.dat* into a new file *dgrid_aims.fhi* with native DGrid file format. This file provides the basis to all capabilities of the DGrid program described in detail in its manual at https://www2.cpfs.mpg.de/~kohout/Documents/dgrid-html/dgrid.html.

`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 diffences 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.

Note: You should no longer need the output species_proj_dos , output atom_proj_dos or output dos at all. Instead, the same objects can be obtained with MUCH better integration accuracy using the alternative keywords output species_proj_dos_tetrahedron , output atom_proj_dos_tetrahedron or output dos_tetrahedron . Please see there for the syntax. The description of the older keywords is kept for now.

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 $\boldsymbol{G}$=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 an improved (so-called perturbative) density of states.

If no dos_kgrid_factors are specified, the original k-point grid is used.

The unit of output for the density of states is $(eV \cdot V_{\text{unit cell}})^{-1}$, i.e., number of states per energy unit (eV) and unit cell volume.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled DOS will have the default filename(s) and the file(s) containing the scalar-relativistic (i.e. no SOC) band structure will have an additional suffix ".no_soc". Note that if you requested `spin collinear` in the `control.in` in addition, the file with spin-orbit coupling included (suffix ".dat") has only one column containing the dos of *all* spin-coupled states. The reason is that the separate spin channels of scalar relativity no longer exist after the SOC operator is applied – the states now form a single set. The output of the total DOS without spin-orbit coupling (suffix ".dat.no_soc") has two columns - one for each spin channel.

---

output **sub-tag:** `dos_tetrahedron` (control.in)

Usage: output dos_tetrahedron Estart Eend n_points

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.

This keyword shares its syntax with output atom_proj_dos_tetrahedron and output species_proj_dos_tetrahedron . See also section 4.4 for more details.

Two output files emerge from this option:

- KS_DOS_total]_raw_tetrahedron.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 $\boldsymbol{G}$=0 component of the long-range Hartree potential (periodic systems).
- KS_DOS_total_tetrahedron.dat contains the same information, except that the energy zero is shifted to the Fermi energy (metallic systems) or valence band maximum (insulators), respectively.

The keyword dos_kgrid_factors is **now** supported by the tetrahedron method.

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 $\boldsymbol{G}$=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.

The unit of output for the density of states is $(eV \cdot V_{\text{unit cell}})^{-1}$, i.e., number of states per energy unit (eV) and unit cell volume.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled DOS will have the default filename(s) and the file(s) containing the scalar-relativistic (i.e. no SOC) band structure will have an additional suffix ".no_soc". Note that if you requested spin collinear in the control.in file in addition, the file with spin-orbit coupling included (suffix ".dat") has only one column containing the dos of *all* spin-coupled states. The reason is that the separate spin channels of scalar relativity no longer exist after the SOC operator is applied – the states now form a single set. The output of the total DOS without spin-orbit coupling (suffix ".dat.no_soc") has two columns - one for each spin channel.

---

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 serial 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 parallel 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 $c_{il}(\boldsymbol{k})$ 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 $c_{il}(\boldsymbol{k})$ 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!) $\chi_{i,k}(\boldsymbol{r})$ as defined through Eq. (22) of the FHI-aims Computer Physics Communications description, Ref. [28].

Note that the output aitranss keyword provides one further option to print eigenvectors and other information for any non-periodic system, serial or parallel (ScaLAPACK, ELPA, ELSI).

This keyword will output scalar-relativistic eigenvectors. For spin-orbit-coupled eigenvectors, please see the soc_eigenvectors keyword.

---

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:** elsi_log (control.in)

Usage: output elsi_log

Purpose: Output ELSI runtime information in a JSON format.

When this flag is enabled, the ELectronic Structure Infrastructure (ELSI) will output information every time it is used to solve the Kohn-Sham eigenvalue problem, including timings for the ELSI invocation, a list of values for relevant variables, and versioning information.

This information is stored in a JSON format and is written to the file elsi_log.json using the FortJSON library, which is distributed with FHI-aims and ELSI and is built automatically. Only task 0 will output this file.

To output information from an FHI-aims calculation in a JSON format, please use the output json_log keyword.

---

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:** h_s_matrices (control.in)

Usage: output h_s_matrices

Purpose: Writes the Hamiltonian and overlap matrices s_{ij} and h_{ij} into separate files. The output format has one line per matrix entry. On this line the first column is the row index of the entry, the second column the column index of the entry and the third column is the value of the entry. Only the upper triangle and the diagonal of the symmetric matrix is written.

This functionality behaves slightly differently in periodic vs. cluster calculations. In the cluster case s_{ij} and h_{ij} are written out in keeping with the packed_matrix_format . In the periodic case only the Gamma-point Hamiltonian is written out and only as a dense matrix regardless of packed_matrix_format settings.

---

output sub-tag: hamiltonian_matrix (control.in)

Usage: output hamiltonian_matrix

Purpose: Writes out the k-point dependent complex hamiltonian matrix for a periodic system for those k-points for which band structure output was requested.

Restriction: For periodic systems only, and KS_method parallel is not supported. Specific band structure output must be requested through output band .

For *periodic* systems, this option allows to write out the k-dependent (Bloch) hamiltonian matrices that correspond to the set of k-points requested with the output band keyword. Both spin channels are written into the same file.

---

output sub-tag: ks_coulomb_integral (control.in)

Usage: output ks_coulomb_integral

Purpose: Writes the Coulomb integrals matrices $< ij|V|kl >$ into the file named coulomb_integrals_mo.out. The output format has one line per matrix entry. On this line the four columns are the indices i, j, k, l, denoting the KS molecular orbitals. The last column is the value of the entry. The whole matrix is written out. This functionality works only for cluster calculations at this point.

---

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 some other 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 $n_{\mathrm{at}}^{\mathrm{free}}(r)$. For the hybrid functionals HSE, PBE0, and B3LYP, the underlying densities used for the partitioning are free-atom PBE and BLYP densities, respectively.

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 the are no rigorously defined boundaries between atoms). Nonetheless, much chemical intuition is based on such a concept.

Hirshfeld's [109] "atoms-in-molecules" partitioning relies on the same idea as the partitioning of our charge density for the electrostatic potential (Eq. 3.19), using the free-atom electron density $n_{\text{at}}^{\text{free}}(r)$ as the weight function $g_{\text{at}}(\boldsymbol{r})$ in Eq. (3.15). 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 [109] "atoms-in-molecules" partitioning is just one among many other prescriptions that have been suggested in the literature. Any ghost atoms in the system are skipped for this analysis. This could lead to "missing" charge if the ghost atoms carried any nonnegligible charge, but that should happen only in abnormal systems.

Note that the `output hirshfeld` keyword itself only writes a Hirshfeld analysis for the final geometry of an FHI-aims run, not, for instance, for intermediate molecular dynamics steps. This ensures that the Hirshfeld analysis does not accidentally clutter an output file with large amounts of data if the `output_level MD_light` output level is set. Output for every geometry can be accomplished with the `output hirshfeld_always` keyword below.

`output` **sub-tag:** `hirshfeld_always` (control.in)

Usage: `output hirshfeld_always`

Purpose: Writes out a Hirshfeld analysis at every geometry step of a run.

This keyword ensures that a Hirshfeld analysis is written at every geometry step of a FHI-aims run, not just after the final step. If `output hirshfeld-I` is requested together with `output hirshfeld_always` , results from both the normal and the iterative Hirshfeld analysis are written at every step.

`output` **sub-tag:** `hirshfeld-I` (control.in)

Usage: `output hirshfeld-I`

Purpose: Produces an "iterative Hirshfeld" analysis of partial charges and moments on each atom.

Similar functionality to the normal `output hirshfeld` Hirshfeld analysis – see the comments for that keyword – except that in the "iterative Hirshfeld" [37] analysis the partition weights are changed. Here, the partitioning densities are not those of neutral atoms but rather those of ions with the same formal charge as the formal charge determined by the "iterative Hirshfeld" analysis.

This keyword is implemented but has not seen much production use. It is therefore not guaranteed that it will always work or that the results will always make sense or even be

in line with the original "iterative Hirshfeld" publication.[37] All may be well, but if you do use the functionality, please check very carefully that the results appear to be correct.

`output` **sub-tag:** `json_log` (control.in)

Usage: `output json_log`

Purpose: Output FHI-aims runtime information in a JSON format.

When this flag is enabled, FHI-aims will output information in a JSON format, which may be easily parsed by your favorite post-processing language (or Python).

This feature was added in 2018, and much of the functionality in FHI-aims outside of the main SCF cycle will not write out any information. A partial list of quantities which will be written includes:

- Initial/final geometries
- Versioning information
- Runtime settings: number of basis functions, number of k-points, etc.
- SCF convergence evolution
- Mulliken decompositions (when calculated)
- Domain decomposition (a.k.a. batch partitioning) statistics
- Total energies
- HOMO/LUMO levels (on the SCF k-grid)
- Fundamental gap (on the SCF k-grid)
- Timings

This information is written to the file `aims.json` using the FortJSON library, which is distributed with FHI-aims and built automatically. Only task 0 will output this file. Unlike the ELSI JSON log (which is written out using the `output elsi_log` keyword), this JSON log will persist through SCF reinitialization, geometry relaxation steps, and MD steps; it will only be overwritten when a new FHI-aims calculation is performed.

To output information from ELSI directly in a JSON format, please use the `output elsi_log` keyword.

`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 $\epsilon_l(\boldsymbol{k})$.

number is an integer number. Default: 1.

Eigenvalues will only be written for the first number k-points by default. For dense $\boldsymbol{k}$-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. This now works if using KS_method parallel.

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 and KS_method serial.

---

output **sub-tag: matrices_2005** (control.in)

Usage: output matrices_2005

Purpose: Writes the Hamiltonian and overlap matrices s_{ij} and h_{ij} into separate files. The output format is the legacy format where the entries of the matrix are written out in five-by-five blocks.

Restriction: This functionality is unavailable for periodic systems, or if a packed_matrix_format is used.

---

output **sub-tag: matrices_elsi** (control.in)

Usage: output matrices_elsi

Purpose: Uses ELSI to output the Hamiltonian and overlap matrices.

Note that the output is written as binary files in the ELSI CSC format (see the documentation of ELSI). The Python script in the "utilities/elsi_matrix" directory may be used to convert an ELSI matrix file to a human-readable format.

---

output **sub-tag: matrices_parallel** (control.in)

Usage: `output matrices_parallel types [format]`

Purpose: Writes distributed 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: `memory_tracking` (control.in)

Usage: `output memory_tracking`

Purpose: Outputs all tracked allocations and deallocations.

Restriction: A large number of allocations and deallocations in FHI-aims are currently not tracked.

`output` sub-tag: `moment_mat_soc` (control.in)

Usage: `output moment_mat_soc`

Purpose: Writes the SOC-perturbed momentum matrices at every k-point of the SCF k-grid to matrix files in the ELSI CSC format. This keyword requires `include_spin_orbit` , `compute_dielectric` , and parallel linear algebra (ScaLAPACK). FHI-aims will stop if this keyword is used in an unsupported case.

Note that the output is written as binary files in the ELSI CSC format (see the documentation of ELSI). The Python script in the "utilities/elsi_matrix" directory may be used to convert an ELSI matrix file to a human-readable format.

`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.

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 the are no rigorously defined boundaries between atoms). Nonetheless, much chem-

ical intuition is based on such a concept.

A classic "atoms-in-molecules" concept is the Mulliken analysis [171], 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.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled Mulliken analysis will have the default filename(s) and the file(s) containing the scalar-relativistic (i.e. no SOC) Mulliken analysis will have an additional suffix ".no_soc".

`output` **sub-tag:** `mulliken_summary` (control.in)

Usage: `output mulliken_summary`

Purpose: Produces a Mulliken analysis of the occupation of each atom and its angular momentum channels (summary only) used.

As above, except here the supplementary `Mulliken.out` is not generated, which can result in substantial savings on filespace.

`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 sto 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:** `onsite_integrands` (control.in)

Usage: `output onsite_integrands`

Purpose: Writes out onsite integrands for all radial functions on the code's internal 'radial' and 'logarithmic' grids.

Since August 2013, FHI-aims verifies the accuracy of its 'radial' integration grid (the sparse grid of atom-centered radial shells around each atom which is part of the definition of its three-dimensional, overlapping atom-centered integration grids) in comparison to integrals on the dense 'logarithmic' grid which is used to set up the spherical free atom, all radial functions etc. in one dimension.

These integrals take the form

$$\int d^3r \phi_i(\boldsymbol{r}) \hat{H} \phi_i(\boldsymbol{r}) = \int dr \, [f(r)] \times \text{angular integral.} \tag{3.199}$$

With our usual definition of basis functions,

$$\phi_i(r) = \frac{u_i(r)}{r} Y_{lm}(\Omega) \quad , \tag{3.200}$$

we get:

$$f(r) = u_i(r) \cdot \left[-\frac{1}{2} u_i''(r) + \frac{1}{2} \frac{l(l+1)}{r^2} u_i(r) + v(r) u_i(r) \right] \tag{3.201}$$

in the non-relativistic case. In the case of scaled ZORA or atomic ZORA scalar relativity, the kinetic energy part is modified and the integrand reads:

$$\begin{aligned} f(r) = u_i(r) \quad \cdot \quad & \left[\frac{2c^2}{2c^2 - v(r)} \cdot \left(-\frac{1}{2} u_i''(r) + \frac{1}{2} \frac{l(l+1)}{r^2} u_i(r) \right) \right. \\ & \left. - \frac{c^2}{(2c^2 - v(r))^2} \cdot v'(r) \cdot \left(u_i'(r) - \frac{u_i(r)}{r} \right) + v(r) u_i(r) \right] . \end{aligned} \tag{3.202}$$

These are the integrands to test both the 'logarithmic' grid and the 'radial' grid around each atom, where $v(r)$ is set to be the one-dimensional potential of a spherical free atom as calculated at the outset of each run.

If `output onsite_integrands` is set to be true, the actual integrands $f(r)$ and various of their parts are printed for each radial function. This is mainly useful for debugging purposes, to understand what we are integrating for a given basis function choice. Especially for contracted Gaussian basis functions, $f(r)$ can look quite ugly near the nucleus.

The files that contain the actual integrand $f(r)$ defined above are called `Onsite_r2_phi_h_phi_log.(function).dat` and `Onsite_r2_phi_h_phi_rad.(function).dat` for the logarithmic and radial grids, respectively, with "(function)" indicating the element and the radial function number in the order used by FHI-aims (for instance, the `output basis` keyword uses the same order to output the radial functions used in the code). Units are Å for the radial coordinate, but atomic units (Ha/bohr^3) for the integrand itself.

`output` **sub-tag:** `overlap_matrix` (control.in)

Usage: output overlap_matrix

Purpose: Writes out the k-point dependent complex overlap matrices for a periodic system for those k-points for which band structure output was requested.

Restriction: For periodic systems only, and KS_method parallel is not supported. Specific band structure output must be requested through output band .

For *periodic* systems, this option allows to write out the k-dependent (Bloch) overlap matrices that correspond to the set of k-points requested with the output band keyword. If output eigenvectors is set in addition, the Kohn-Sham eigenvectors $c_{il}(\boldsymbol{k})$ will be written into separate files for each spin channel, *only* for each k-point for which band output was requested.

---

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 serial.

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: postscf_eigenvalues** (control.in)

Usage: output postscf_eigenvalues

Purpose: For periodic systems, writes all Kohn-Sham eigenvalues on a potentially dense k-space grid to an ASCII file 'Final_KS_eigenvalues.dat'.

Restriction: Works only for periodic systems. Does not work when keyword use_local_index is set.

If this keyword is set, the eigenvalues and occupation numbers for a periodic system are recomputed and written to a file 'Final_KS_eigenvalues.dat' after the s.c.f. calculation (and, possibly, relaxation, dynamics etc.) is complete. A denser k-space grid than during the original s.c.f. calculation can be specified using the dos_kgrid_factors keyword.

Note that the resulting output file can become very large. See the header of the 'Final_KS_eigenvalues.dat' for details and for units used.

Note that this additional calculation is done using serial solutions of the eigenvalue problems for individual k-points on individual CPU cores. This always works but will create memory problems as the system size increases, simply because local copies of all matrices are kept on single CPUs. For large systems, our usual, more sophisticated parallelization strategies have not yet been copied over to this routine.

Finally, note that an externally generated k-point list k_list.in (see keyword

k_points_external) is not supported by output postscf_eigenvalues and an internally generated, even-spaced k-point grid (also defined in k_list.in) is used instead.

---

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_and_derivs_on_grid (control.in)

Usage: output rho_and_derivs_on_grid

Purpose: Writes the post-processing density and derivations(including Laplacian) evaluation in FHI-aims.
Several notes
===
The definition of weight (called partition_tab in aims code) of the integration point is in the line after the eqn (20) from Computer Physics Communications 180 (2009) 2175-2196
And also, we have this equation
$\sum_i w(r_i)\rho(r_i) = n_{electron}$
===
The density gradient is output as a vector. Post-processing is needed if you need sigma or something else.
===
The kinetic energy density tau is defined w/o the 1/2 coefficient.
$\tau(\boldsymbol{r}) = \sum_{ij} \nabla\varphi_i(\boldsymbol{r}) n_{ij} \nabla\varphi_j(\boldsymbol{r})$
===
We don't evaluate properties on those points with zero weight (partition_tab). I still print them out but those points have all properties equal zero. You might want to throw them away before usage.
===
For spin polarized calculation, two files are outputted (spin 1 and spin 2).
===

---

output sub-tag: rho_multipole (control.in)

Usage: output rho_multipole

Purpose: Writes the partitioned atom-centered charge multipole components $\delta\tilde{n}_{\mathrm{at},lm}(r)$ to individual files for each atom, l, and m (see Eq. 3.19).

---

output sub-tag: soc_eigenvalues (control.in)

Usage: output soc_eigenvalues

Purpose: Writes the SOC-perturbed eigenvalues at every k-point of the SCF k-grid to an output file named SOC_eigenvalues.dat. This keyword will not enable spin-orbit coupling; if spin-orbit coupling is not enabled via the include_spin_orbit keyword, this keyword will be ignored.

---

output sub-tag: soc_subspace_in_band (control.in)

Usage: output soc_subspace_in_band

Purpose: Writes the spin-orbit coupled eigenvectors in a basis of non-spin-orbit coupled eigenstates (e.g., in the basis formed by the scalar-relativistic eigenstates of a self-consistent, non-spin-orbit-coupled calculation). This is only done for k-points included in any band structure segments requested to be written out.

This keyword must be used together with include_spin_orbit and output band or band_mulliken .

FHI-aims normally solves the non-selfconsistent spin-orbit coupled eigenvalue problem (including eigenvectors) in a basis of self-consistent eigenvectors calculated in a preceding, scalar-relativistic calculation (no spin-orbit coupling). This means that the spin-orbit coupled eigenvectors are immediately available and can be interpreted in terms of their origin from an underlying scalar-relativistic basis set.

The spin-orbit coupled eigenvectors in terms of their non-spin-orbit-coupled counterparts will be written for each sampled k point in each specified band in control.in.

As an example, assume that there are n scalar-relativistic (i.e.,pre-SOC) eigenstates in a closed-shell (non-spinpolarized) self-consistent calculation. After the SOC perturbation is applied and the resulting eigenvalue problem solved, this leads to $2n$ spin-orbit coupled eigenvectors. These are the columns of a ($2n \times 2n$) matrix, where the first n rows represent n pre-SOC states of the scalar-relativistic spin channel 1 and the following n rows (row numbers (n+1) to $2n$) represent the n pre-SOC states of the scalar-relativistic spin channel 2. Since the original scalar-relativistic eigenvectors form an orthonormal basis set, the absolute value of element (i,j) represents the contribution from the pre-SOC state corresponding to the ith row to the post-SOC state corresponding to the jth column.

Note that for parallel calculations, the output is written as binary files, using the ELSI infrastructure (for details, the documentation of ELSI covers this format). You may want to use the script convert_elsi_to_mm.py together with read_elsi.py in the utilities/elsi_matrix directory to create a human-readable format.

---

output sub-tag: species_proj_dos (control.in)

Usage: output species_proj_dos Estart Eend n_points broadening

Purpose: Writes a projected, angular-momentum resolved partial density of states (pDOS), adding up the contributions of 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.

Note: You should no longer need the output species_proj_dos , output atom_proj_dos or output dos at all. Instead, the same objects can be obtained with MUCH better integration accuracy using the alternative keywords output species_proj_dos_tetrahedron , output atom_proj_dos_tetrahedron or output dos_tetrahedron . Please see there for the syntax. The description of the older keywords is kept for now.

Different from the atom_proj_dos option, this option adds up the pDOS contributions of all atoms of each species defined in control.in and used in geometry.in. This provides a quick handle to obtain the pDOS contribution of 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 te 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.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled DOS will have the default filename(s) and the file(s) containing the scalar-relativistic (i.e. no SOC) DOS will have an additional suffix ".no_soc". Note that if you requested `spin collinear` in the `control.in` file, there will be only *one* file per species (and equivalently for all `_raw.dat` files) containing the projected DOS of all spin-coupled states:

- *species*`_l_proj_dos.dat`

The reason is that the separate spin channels of scalar relativity no longer exist after the SOC operator is applied – the states now form a single set. However, there are two DOS files for the output without spin-orbit coupling; one for each spin channel:

- *species*`_l_proj_dos_spin_up.dat.no_soc`
- *species*`_l_proj_dos_spin_dn.dat.no_soc`.

`output` **sub-tag: `species_proj_dos_tetrahedron`** (control.in)

Usage: `output species_proj_dos Estart Eend n_points`

Purpose: Writes an projected, angular-momentum resolved partial density of states (pDOS) adding up the contributions of 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.

This option is based on a Mulliken Analysis and shares its syntax with `output dos_tetrahedron` and `output atom_proj_dos_tetrahedron` . See also section 4.4 for more details.

Different from the `atom_proj_dos_tetrahedron` option, this option adds up the pDOS contributions of all atoms of each `species` defined in `control.in` and used in `geometry.in`. This provides a quick handle to obtain the pDOS contribution of 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_tetrahedron_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 $\boldsymbol{G}$=0 component of the long-range Hartree potential (periodic systems).

- *species*_l_proj_dos_tetrahedron.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 te 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.

This keyword supports spin-orbit coupling. When spin-orbit coupling is enabled, the file(s) containing the spin-orbit-coupled DOS will have the default filename(s) and the file(s) containing the scalar-relativistic (i.e. no SOC) DOS will have an additional suffix ".no_soc". Note that if you requested `spin collinear` in the `control.in` file, there will be only *one* file per species (and equivalently for all `_raw.dat` files) containing the projected DOS of all spin-coupled states:

- *species*_l_proj_dos_tetrahedron.dat

The reason is that the separate spin channels of scalar relativity no longer exist after the SOC operator is applied – the states now form a single set. However, there are two DOS files for the output without spin-orbit coupling; one for each spin channel:

- *species*_l_proj_dos_tetrahedron_spin_up.dat.no_soc
- *species*_l_proj_dos_tetrahedron_spin_dn.dat.no_soc.

`output` sub-tag: `v_eff` (control.in)

Usage: `output v_eff`

Purpose: Writes the local effective potential $v_{\text{eff}}(\boldsymbol{r})$ at each integration grid point $\boldsymbol{r}$ 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. [28]). 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 $\delta\tilde{v}_{\mathrm{at},lm}(r)$ 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`).

3.42 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_method` s 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 $\delta\tilde{v}_{\mathrm{at},lm}(r)$ 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 I.2), 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 benefet over `calc_hartree`. Note that the (legacy) keyword `distributed_spline_storage` should be false for `shmem`, at least.

Tag: `distribute_leftover_charge` (control.in)

Usage: `distribute_leftover_charge .true./.false.`
This keyword is superseded by the `compensate_multipole_errors` keyword which solves the problem of small residual charge integration errors in slab calculations much better than `distribute_leftover_charge` .

Keyword `distribute_leftover_charge` could introduce noticeable total energy inaccuracies especially for "light" integration grid settings.

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. [28] 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. [96] 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

Also, be sure to check what the Kohn-Sham eigenvalues mean if you need them, do not use them blindly. `multiplicity` shifts the eigenvalues!

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 capabilitues 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_2O molecule towards its minimum-energy structure. The relevant input files `geometry.in` and `control.in` are included in the directory *testcases/H2O-relaxation* (see Chapter 1).

The starting geometry here is badly distorted H_2O, 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^{-2} 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 trememdous 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 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^{-2} 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, do not simply use a setting of 10^{-4} eV/Å because it feels more accurate – it will only end up probing some irrelevant numerical traits of the energy surface, 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 [155], see keyword `init_hess` .

The `control.in` could also include other general keywords concern the technicalities of obtaining self-consistency, which are not mandatory and are therefore not included in this simple test case. Examples include a broadening of occupation numbers around the Fermi level using the `occupation_type` keyword (this has no physical impact in a molecule with a HOMO-LUMO gap but may be important in metals), a mixing parameter `charge_mix_param` for the `mixer` (nonlinear optimization of the s.c.f. cycle), and convergence criteria for the s.c.f. cycle. FHI-aims attempts to choose reasonable settings for these aspects automatically in order to help avoid lengthy mistakes. The s.c.f. criterion for density convergence, `sc_accuracy_rho` in particular is set tightly by default, in order to have forces that are already mostly converged when the (more expensive) force computation is first done.

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 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.
- Reinitialization, s.c.f., and geometry optimization information is repeated for each successive geometry step until the optimum geometry is found within the force tolerance specified by `relax_geometry` .

Key information that should be paid attention to is the following:

Eigenvalue and total energy information:
This should be checked for roughly realistic total energy values, a reasonable eigenvalue spectrum: Make sure that the expected core states, the correct number of eigenstates, and possibly the expected HOMO-LUMO gap are all in place.

As in all other iterations, *three* variants of the total energy are given (here quoted from the *initialization* step, just prior to the start of the s.c.f. cycle for the initial geometry):

```
| Total energy                  :          -76.44577378 Ha        -2080.19534383 eV
| Total energy, T -> 0          :          -76.44577378 Ha        -2080.19534383 eV
| Electronic free energy        :          -76.44577378 Ha        -2080.19534383 eV
```

For systems with a HOMO-LUMO gap, these values should all be the same, but for systems with fractional occupation numbers (metallic systems, large `occupation_type` setting, or degenerate levels at the Fermi level), this is not generally the case.

For molecular systems, the meaning of fractional occupation numbers is questionable, and we recommend to always rely on the first value, the "Total energy".

For metallic systems, fractional occupation numbers are unavoidable. However, it is possible to estimate the total energy for zero occupation broadening ("Total energy, T $->$ 0") based on an entropy expression associated with the fractional occupation numbers. This estimate is based on a free electron gas argument; do not trust it for atoms or small molecules. Bulk metals or metallic surfaces benefit from this correction, and their total energy may be estimated from the extrapolated total energy ("Total energy, T $->$ 0").

The "free energy" simply sums the total energy together with the full entropy correction from the fractional occupation numbers. This quantity has no physical meaning, but it is strictly *this* free energy to which our calculated forces correspond. The reason is that fractional occupation numbers may carry an implicit derivative of their own with respect to the nuclear coordinates. This would enter the total energy, but this derivative is not available in simple analytical terms in FHI-aims.

Geometry information:
After each relaxation step, the convergence of the present geometry is checked (by monitoring the maximum remaining force component on any atom in the structure), and

the updated geometry to be treated next is printed. For instance, the final (converged) geometry can be found near the end, in a format that is directly suitable for a follow-up run:

```
------------------------------------------------------------
  Final atomic structure:
                        x [A]             y [A]             z [A]
           atom         0.00000000       -0.07327020       -0.00000000  O
           atom         0.76741277       -0.67036490        0.00000000  H
           atom        -0.76741277       -0.67036490        0.00000000  H
------------------------------------------------------------
```

By default, a version of this information is also written into a file `geometry.in.next_step`, which should be used (together with the `hessian.aims` file) to continue a relaxation run and also to begin a possible post-relaxation with *tight* settings (the `hessian.aims` file contains a usually much better Hessian for the structure at hand than the starting guess).

Timing information:
Finally, we note that FHI-aims also provides detailed timing information as well as partial memory accounting 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     :  20200114, Time     :  104524.650

          Computational steps:
          | Number of self-consistency cycles          :           49
          | Number of SCF (re)initializations          :            5
          | Number of relaxation steps                 :            4

          Detailed time accounting                     :  max(cpu_time)    wall_clock(cpu1)
          | Total time                                 :        3.228 s           3.255 s
          | Preparation time                           :        0.067 s           0.067 s
          | Boundary condition initalization           :        0.000 s           0.000 s
          | Grid partitioning                          :        0.041 s           0.041 s
          | Preloading free-atom quantities on grid    :        0.028 s           0.028 s
          | Free-atom superposition energy             :        0.047 s           0.048 s
          | Total time for integrations                :        1.394 s           1.404 s
          | Total time for solution of K.-S. equations :        0.032 s           0.030 s
          | Total time for EV reorthonormalization     :        0.001 s           0.001 s
          | Total time for density & force components  :        0.986 s           0.991 s
          | Total time for mixing                      :        0.063 s           0.068 s
          | Total time for Hartree multipole update    :        0.050 s           0.049 s
          | Total time for Hartree multipole sum       :        0.400 s           0.402 s
          | Total time for total energy evaluation     :        0.008 s           0.014 s

          [...]
```

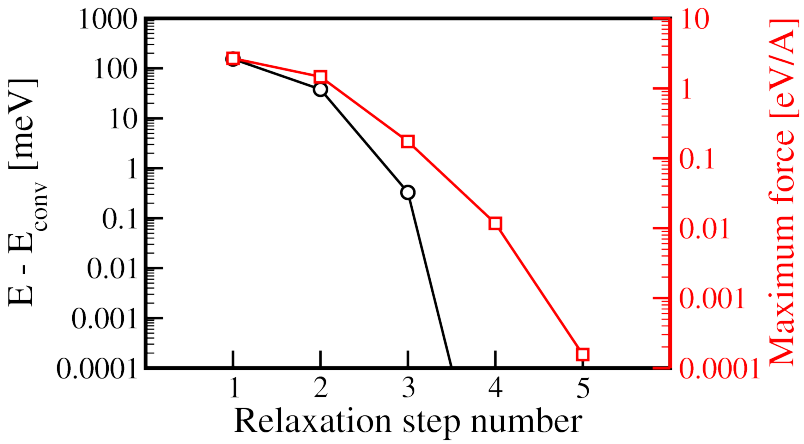


Figure 4.1: Total energy convergence (left y axis) and force convergence (right y axis) during the relaxation test run of H_2O. A total of five steps is taken (initial geometry plus four relaxation steps). The total energy of the final step is taken as the reference here, and the total energy in the 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.

```
          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 104524 s, but rather *ended* at 10:45:24 h, one fine January 14.

In addition, detailed timing is provided both for as elapsed CPU time (on individual CPUs during the run), and actual elapsed wall clock time. In a normal production run, no significant discrepancies should occur between wall clock and CPU times. If discrepancies arise, they could indicate serious problems with load balancing or communication in a parallel run.

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 follows:

```
> 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`.

To continue a relaxation run (for instance, for a post-relaxation step with tight settings, see below), use the files `geometry.in.next_step` and `hessian.aims` that are created by default during a relaxation. They contain the atomic structure of the current relaxation step and the current estimate of the Hessian matrix, respectively, as created by the `bfgs` relaxation algorithm. This output can be fine-tuned (or prevented) using the `write_restart_geometry` and `hessian_to_restart_geometry` keywords.

Obviously, all intermediate geometries (but not the Hessian) are also part of the standard output stream by default, in the right format for a restart.

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_2O 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

Actually, this section could easily apply to all elements. As outlined in Section 3.8, it seems that the `relativistic atomic_zora scalar` keyword and the underlying "atomic ZORA" approach as implemented in FHI-aims (Equations (55) and (56) of Ref. [28]) perform on par with the best available scalar-relativistic benchmark methods (see Refs. [152, 113]). Just use this approach in production, unless there is a specific reason not do do so (a few methods in FHI-aims that are still being implemented may initially only support a non-relativistic treatment).

Note that total energy *differences* will be very large between different relativistic treatments, so it is best to stick to a single level of scalar relativity for all calculations.

With H and O, the simple H_2O testcase of the preceding Section 4.1 involves only elements so light that relativistic effects on their total energies can still be 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 reqired 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. [28], 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. [195]) – but bear in mind that relativistic effects should not simply be shrugged off!

As noted in Section 3.8, we recommend "atomic ZORA" as the blanket default where possible, invoked by the keyword

```
relativistic atomic_zora scalar
```

Additionally, the effects of spin-orbit coupling may be included in energy levels / band structures, densities of states, etc. post-selfconsistently, i.e., after the self-consistent calculation is complete. The keyword to do this is:

```
include_spin_orbit
```

That's it.

We illustrate the practical use of atomic ZORA and spin-orbit coupling for the Au dimer in DFT-LDA, found in the *testcases/Au_dimer* directory. Importantly, this follows the always recommended two-step approach for relaxations: First, a *light* prerelaxation (saves much time for steps of the relaxation algorithm that will be completely irrelevant for the final result) and second, a *tight* post-relaxation for final results.

The testcase here uses semilocal DFT, which is relatively cheap. For the much more costly hybrid density functionals, *tight* settings can be prohibitively expensive, and *intermediate* settings (where available) are often completely sufficient.

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=2.464 Å. (The difference to the *tight* result below is quite minor – 0.012 Å.)

After this relaxation run is complete, a file `geometry.in.next_step` is written out by FHI-aims. This file contains the final, relaxed geometry from the "light" prerelaxation run just performed, as well as the Hessian matrix estimate created by the `relax_geometry bfgs` relaxation algorithm. This file can serve as an improved guess for `geometry.in` in the next, typically more expensive "tight" post-relaxation step.

In the subdirectory *postrelax_tight*, 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 (without the h function) used here for Au is quite sufficient for an accurate result. The binding distance converges after two additional relaxation steps, at d=2.452 Å.

We also added in spin-orbit coupling for completeness using the `include_spin_orbit` keyword.

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 Γ-centered grid,
- the grid suggested by Monkhorst and Pack [170] and
- an off-Γ grid.

They are defined as

$$u_r = \begin{cases} \frac{r-q}{q} & \Gamma\text{-centered}, \\ \frac{2r-q-1}{2q} & \text{Monkhorst-Pack}, \\ \frac{r-1}{q} + \frac{1}{2q} & \text{off-}\Gamma \end{cases} \tag{4.1}$$

where $r = 1 \ldots q$ 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[1] agrees with the Γ-centered grid for an uneven number of grid points and with the off$-\Gamma$ grid for an even number of points. The Γ-centered grid and sequentially the Monkhorst-Pack grid with an uneven number of grid points include the Γ-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-Γ 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).

$$\text{k\_offset} \begin{cases} 0.0 \quad 0.0 \quad 0.0 & \Gamma\text{-centered}, \\ \frac{1}{2} - \frac{1}{2n_1} \quad \frac{1}{2} - \frac{1}{2n_2} \quad \frac{1}{2} - \frac{1}{2n_3} & \text{Monkhorst-Pack}, \\ \frac{1}{2n_1} \quad \frac{1}{2n_2} \quad \frac{1}{2n_3} & \text{off-}\Gamma \end{cases} \tag{4.2}$$

[1] One could argue that Monkhorst and Pack never intended their formula to be used for for odd q. Still, for the sake of this section, "Monkhorst-Pack" refers to this definition.

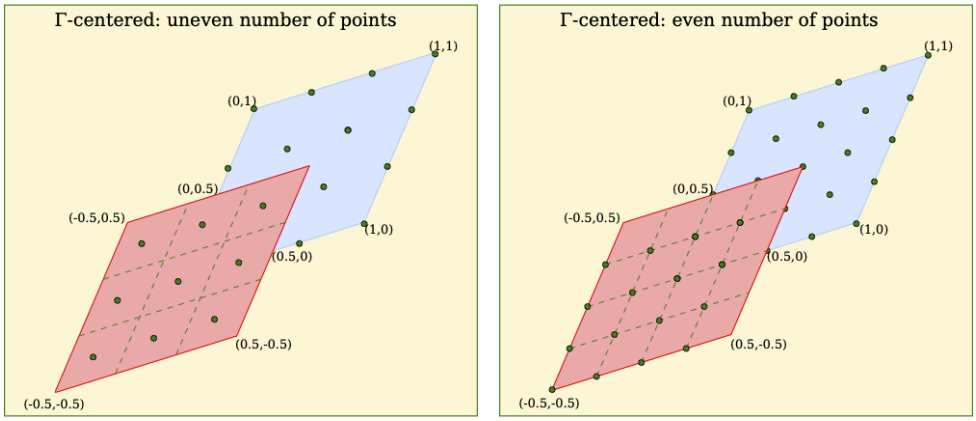


Figure 4.2: The Γ-centered grid in two dimensions for even and uneven numbers of grid points. In both cases the Γ-point is included in the grid.

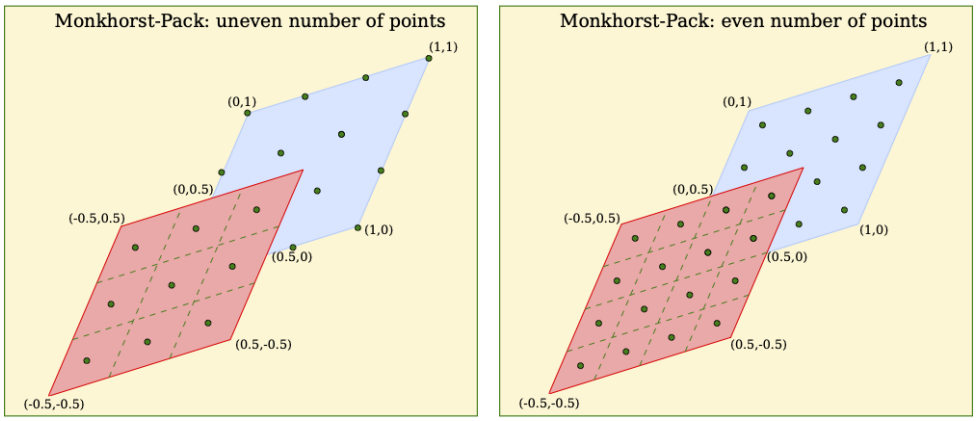


Figure 4.3: The Monkhorst Pack grid [170] in two dimensions for even and uneven numbers of grid points. In the left picture (uneven number of grid points) the Γ point is included in the grid.

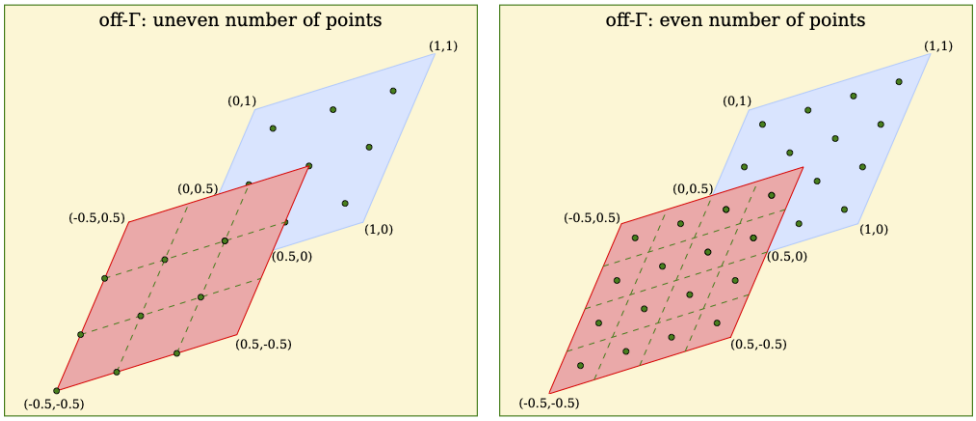


Figure 4.4: The off-Γ grid in two dimensions for even and uneven numbers of grid points.

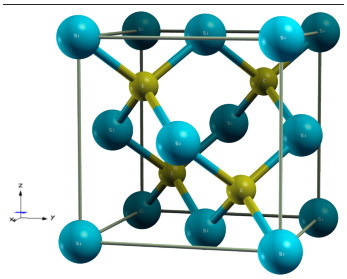


Figure 4.5: The crystal structure of silicon carbide in its cubic zinc blende structure (3C-SiC)

Here,we present a concrete example. Cubic silicium carbide (3C-SiC) is a semicoductor featuring an indirect band gap between the Γ-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 $4.36\,\text{Å}$ [118].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.

The oscillatory behaviour in Fig. (4.6) of the Monkhorst-Pack grid reflects the agreement with the off-Γ grid in the case of an even number of grid points and the Γ-centered for uneven number of grid points respectively.

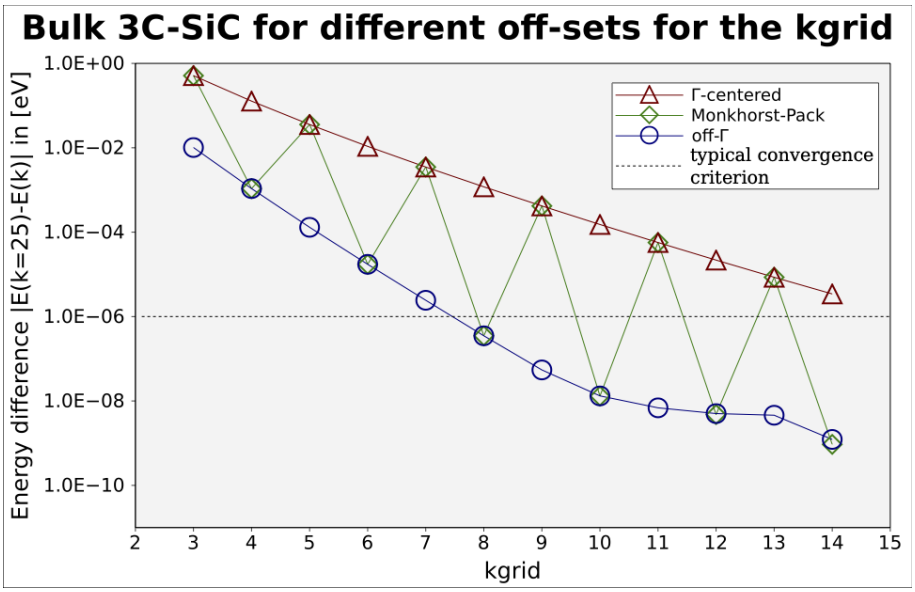


Figure 4.6: The k-grid was tested with respect to the grid density and the shift of the grid in reciprocal space. The shifts are defined in Eqn. (4.1).

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-Γ shift the calculation is well converged with a k-grid of $8\times8\times8$. For a Γ-centered grid the calculation did not reach convergence with a grid of $14\times14\times14$. As mentioned before, in 3C-SiC the VBM is in the Γ-point. Therefore the convergence of the total energy with respect to the density of the k-grid is slow for every grid including the Γ-point, because the contribution of the energy in the Γ-point is over-weighted. In this case the off-Γ shift gives best convergence.

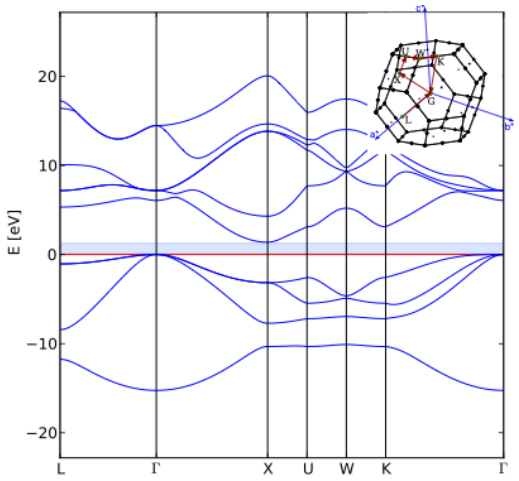


Figure 4.7: The band structure of 3C-SiC with tight, Tier 2 basis settings.

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.

4.4 Plotting the band structure and density of states of a solid

An example for plotting band structures and the density of states is contained in the example for bulk silicon (diamond lattice) in the testcases directory. Likewise, the fcc Al test case contains the necessary lines for control.in (just commented out).

This section contains a walk-through of the necessary steps and keywords to obtain this band structure and density of states. First, let us look at the relevant lines in `control.in` as given in the test case.

```
# output DOS
  output dos_tetrahedron -18.  0.  200

# output band structure
  output band 0.5   0.5   0.5    0.0   0.0   0.0    50  L     Gamma
  output band 0.0   0.0   0.0    0.0   0.5   0.5    50  Gamma X
  output band 0.0   0.5   0.5    0.25  0.5   0.75   50  X     W
  output band 0.25  0.5   0.75   0.375 0.375 0.375  50  W     K
```

To get an output of the band structure, use the keyword `output band`. Each 'band' line covers a single line in reciprocal space, from a starting point to an end point and with a given number of points inbetween.

The total density of states of the system is obtained by the 'output dos_tetrahedron' line, from a starting single-particle energy to an ending single-particle energy, with a specified number of points inbetween to create a smooth line (see section 3.41 for details). The tetrahedron here indicates the implementation of tetrahedron method to achieve better integration accuracy.

The density of states of a solid is a continuous function, but originates from a Brillouin zone integral,

$$g(\epsilon) = \sum_n \int_{\mathrm{BZ}} d^3k \, \delta(\epsilon - \epsilon_n(\boldsymbol{k})) \, . \tag{4.3}$$

The index n runs over all bands, and $\boldsymbol{k}$ is the crystal momentum.

As it stands, this expression for the density of states creates a problem. We obtain it from a discrete set of k-points, rather than as a continuous integral. The density of states from a discrete k-mesh would thus be a sequence of δ-function peaks.

With the lines for band structure output, DOS output_tetrahedron in `control.in`, we can run the example calculation in the testcases directory.

Once this is done, a number of new files have appeared (in addition to the usual standard output stream): The files `band1XXX.out`, `KS_DOS_total_tetrahedron.dat`, and `KS_DOS_total_raw_tetrahedron.dat`.

These files contain the necessary information in column formats that can be manipulated for further output with standard plotting tools. The result of such a manipulation is shown in Fig. 4.8.

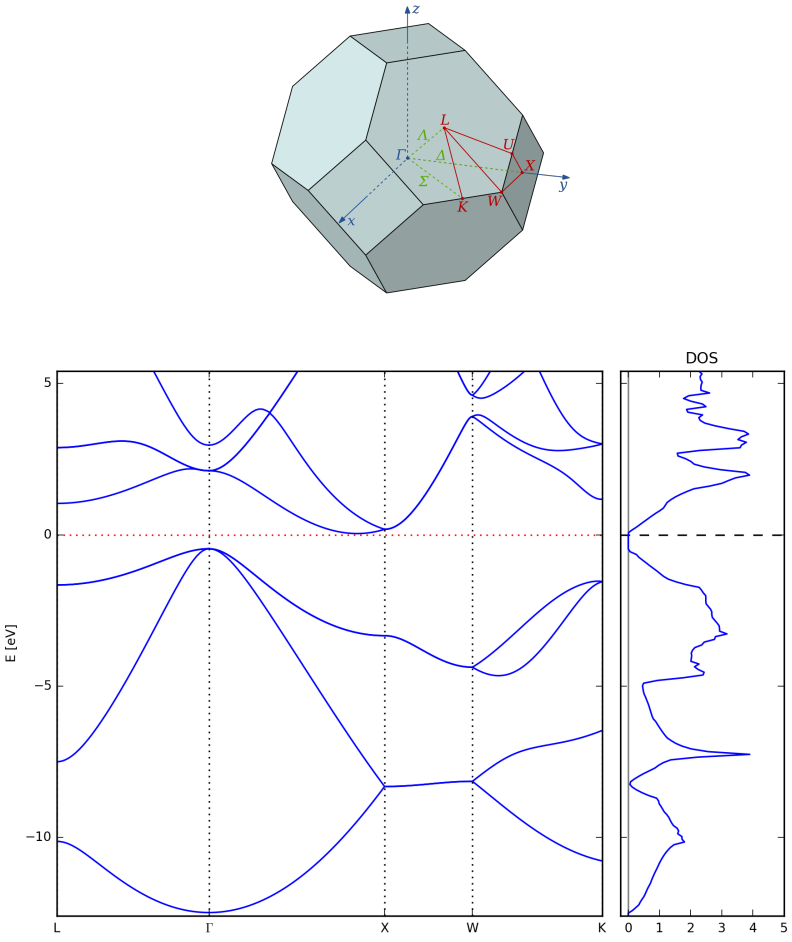


Figure 4.8: Band structure and density of states of the bulk Si test case as generated by the `aimsplot.py` script, which is located in the "utilities" directory. A schematic visualization of the Brillouin zone of the fcc Bravais lattice including some high-symmetry k-points is also shown (taken from Wikipedia).

However, it would be nice to not have to personally massage the output data every time to get this kind of output. The script 'aimsplot.py', located in the utilities directory, should do this for you (type "`aimsplot.py --help`" for more information regarding available options). Simply run it in the output directory, and a version of the band structure and DOS should automatically appear as a png file.

Note: You will need python3 (not python2) to run `aimsplot.py` and you will need a compatible version of `matplotlib` (a common python library) installed, or else the `aimsplot.py` script will exit with a verbose but (to the uninitiated) cryptic error message.

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 $\boldsymbol{G}$=0 component of the long-range

electrostatic potential).

The file KS_DOS_total_tetrahedron.dat contains the same information, but the energy zero is shifted to the Fermi-level of the system. Note that, for metals, the Fermi level is usually well defined. For insulators, it is not well defined, and may end up anywhere in the gap (at some location where FHI-aims finds the normalization criterion for the number of electrons to be sufficiently fulfilled).

If, for a semiconductor, you wish to have the energy zero of a DOS shifted to the top of the valence band, you will have to do this as an extra step. Identify the location of the valence band edge (highest occupied level) either from the FHI-aims output or from the band structure, and shift the x-axis of KS_DOS_total_tetrahedron.dat and other files by the appropriate amount.

Except for the *total* density of states, there are also a few other options to obtain densities of states with some local, spatial resolution. In particular:

- an atom-projected density of states, using output atom_proj_dos_tetrahedron
- an angular-momentum component projected density of states averaged over all atoms *of a single species*, using the keyword output species_proj_dos_tetrahedron . 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 species-projected density of states (but not currently the atom-projected version) will also be automatically plotted by aimsplot.py.

Note that there are two useful projections of the DOS, both of which are based on a Mulliken analysis. output species_proj_dos_tetrahedron can be used to get a species-dependent angular momentum projection, while output atom_proj_dos_tetrahedron resolves the states into the various atomic angular-momentum contributions.

As mentioned above, the species_proj_dos_tetrahedron keyword can also be used to get the DOS contribution from a whole group of atoms at once. Just duplicate the species defaults for the element(s) in question in control.in, give the species a new name, and then use that new species (say) for all surface atoms in geometry.in. This will get you the density of states contribution from all surface atoms.

Finally, an element-decomposed version of the band structure can also be obtained, using the output band_mulliken keyword as an alternative. Two separate scripts for visualization are available in the utilities directory, band_mlk.py and band_mlk_soc.py, for non-SOC and SOC, respectively. Like aimsplot.py, these scripts are intended as *starting points* for plotting purposes and should be customized further, if necessary. The python matplotlib library is used to generate these plots and documentation of this package is available online.

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 3.41, specifically the `output` keyword and the `cube` subkeyword described there). The visualization described in this section pertains to a simple H_2O molecule, and can be repeated for example for the relaxed geometry obtained in the relaxation testrun for H_2O in this distribution.

After running FHI-aims for the desired target geometry with the appropriate output flags (see Section 3.41), some files will be generated, all of which have the extension "*.cube". In the H_2O 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 [123] is presented.

The CUBE files are composed of a header and the volumetric data. The header is divided in the following manner:

- 1^{st} and 2^{nd} lines: text ignored by visualization programs
- 3^{rd} line: number of atoms in the file followed by the origin around which the volumetric data will be plotted.
- 4^{th}, 5^{th}, and 6^{th} lines: number of points to be plotted, followed by the axis vector.
- 7^{th} 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 [123], 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:

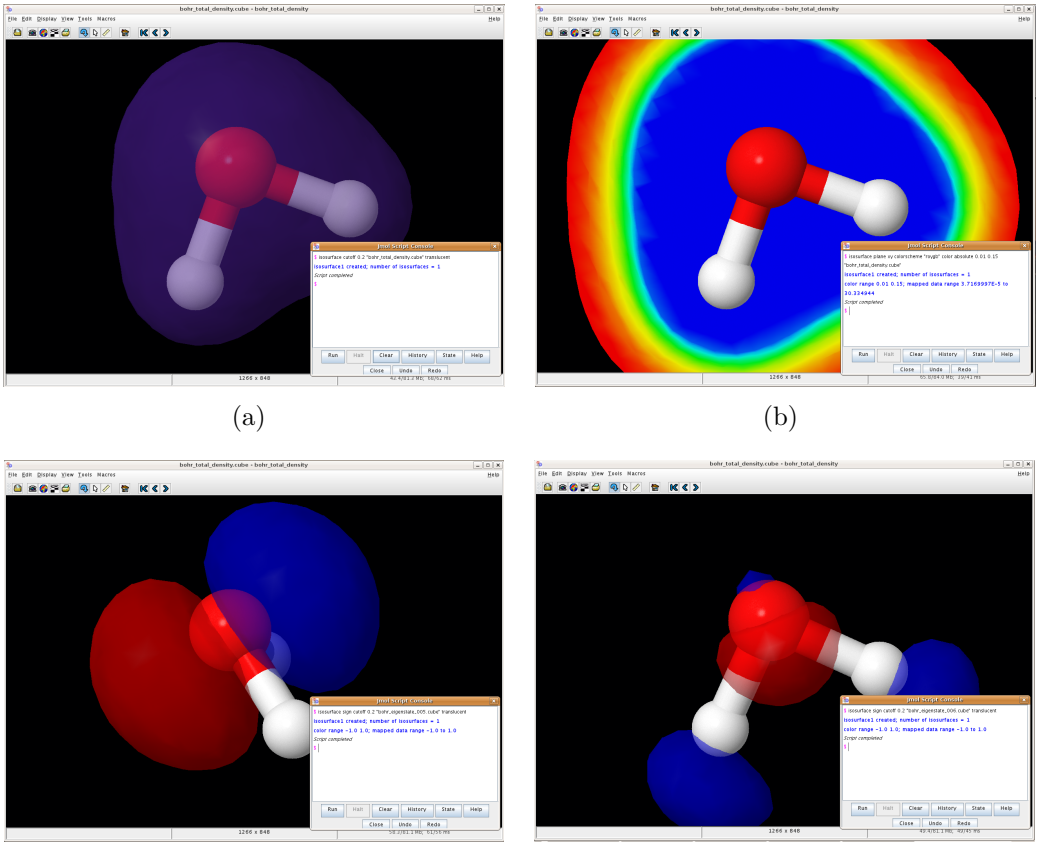
```
> isosurface plane <plane_position> <other_options> "total_density.cube"
```

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 4.9. The CUBE files shown were all converted to atomic units.



(a) (b)

(c) (d)

Figure 4.9: Jmol plots of charge densities and orbitals.

4.6 Computation of vibrational and phonon properties

Vibrations (non-periodic structures) and phonons (periodic structures) can be computed in FHI-aims within different approaches. On the one hand, there is the finite-difference approach. On the other hand, vibrational and phonon properties are accessible through density-funtctional prturbation theory (DFPT). Note that currently the DFPT part is still considered experimental as, e.g., not all XC functionals are supported. Thus, for production calculations we recommend to use the finite-difference methods described below. (Or you know what you are doing. However, cross-checks with finite-difference methods are recommended anyway.)

Currently, vibrational frequencies via the finite-difference method can be calculated in two ways:

4.6.1 Perl script: aims.vibrations.*.pl (non-periodic systems)

There is an older perl script and source codes for this step for the entire finite-difference calculation included with FHI-aims. To compile the tools with CMake, use the target name `vibrations`, e.g., `make -j vibrations`. Location of the Perl script for running the calculations is shown at the end of the make output. Otherwise, when using Makefile and its backend, change into the *src/* directory, from which you also compiled the original FHI-aims. With the same Makefile settings as for the main program, type either `make vibrations`, `make vibrations.mpi`, or (in most cases) `make vibrations.scalapack.mpi`.

For the FHI-aims vibrations target, we recommend the `vibrations` target with CMake and the `vibrations.scalapack.mpi` target with Makefile. Here is an example initial cache file to use with CMake:

```
set(CMAKE_Fortran_COMPILER "mpif90" CACHE STRING "")
set(CMAKE_Fortran_FLAGS "-O3 -ip -fp-model precise" CACHE STRING "")
set(Fortran_MIN_FLAGS "-O0 -fp-model precise" CACHE STRING "")
set(CMAKE_C_COMPILER "icc" CACHE STRING "")
set(CMAKE_C_FLAGS "-O3 -ip -fp-model precise -std=gnu99" CACHE STRING "")
set(LIB_PATHS "/opt/intel/mkl/lib/intel64" CACHE STRING "")
set(LIBS "mkl_intel_lp64 mkl_sequential mkl_core
mkl_blacs_intelmpi_lp64 mkl_scalapack_lp64" CACHE STRING "")
```

This 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 the vibrations are also parallel.

Having compiled the vibrations script, running it is very simple. We use the same water molecule that was relaxed in the example section 4.1. This test case is contained in the directory `testcases/H2O-vibrations`.

We are here calculating a second derivative of the energy from the numerical change on the forces arising from small finite displacements δ, in the following way:

$$\frac{\partial E}{\partial x_i \partial x_j} = \frac{\boldsymbol{F}_i(x_j + \delta) - \boldsymbol{F}_i(x_j - \delta)}{2\delta} \tag{4.4}$$

Here, E is the total energy as a function of all atomic coordinates x_i (i=1,...,3M for M individual atoms), and $\boldsymbol{F}_i$ are the components of the analytical first derivatives (forces) on each atom (again, i=1,...,3M) due to a displacement of coordinate another x_j by an amount plus or minus δ.

Since we are interested in the Hessian in a local minimum of the potential energy surface (ideally, $\boldsymbol{F}_i(\{x_j\})$=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 δ, 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 δ, 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 δ 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 δ 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 δ) 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 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 strucure.

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 δ, 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 δ multiple times in the same directory, starting from the same input files.

The second parameter, `delta`, is the finite displacement δ 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^(-1)]	Zero point energy [eV]	IR-intensity [D^2/Ang^2]
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 δ=0.001 Å produces the following results:

Mode number	Frequency [cm^(-1)]	Zero point energy [eV]	IR-intensity [D^2/Ang^2]
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 δ=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 δ can in fact inflate any residual noise. So, a small value δ=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 δ, either way. This is generally true, and would even remain true for a yet larger value δ=0.005 Å, which would also be a reasonable default choice.

Finally, a word about restarting the `aims.vibrations.*.pl` script if it performed a part of the needed single-point force calculations but did not finish all of them, typically as a result of queue time limits on a shared computer. In short, one can simply restart the script in the same directory as before right away. The script will figure out how far it got during its last run and it will then pick up from there to continue and eventually finish the calculation of vibrational frequencies and free-energy terms.

Raman spectra: A third optional argument is available in the vibrations script, which allows for the calculation of harmonic Raman spectra. In order to obtain such spectra, one should type the following,

`aims.vibrations.*.pl ⟨jobname⟩ delta polar .`

The polarizabilities are obtained with DFPT, and the derivatives with respect to the displacement are then calculated in a similar way as for IR spectra. All the formulae can be found in the following reference: Neugebauer et al., J. Comput. Chem. 23: 895-910, 2002. In particular, Eq. (43) is used to produce the final Raman intensities. Keep in mind that Raman spectra obtained in such fashion are much more time-consuming than IR spectra, the latter requiring only DFT. Note that Raman spectra for solids can also be calculated in this manner (only Γ-point is involved). The Raman intensities that are outputted from this script correspond to combinations of the different tensor components combined to form what is commonly called "powder" spectrum.

In the following we provide the description of tags defining the output of free energies.

Tag: `vibrations` (control.in)

Usage: `vibrations [subkeywords and their options]`

Purpose: allows to choose temperature/pressure range for output of free energies. It has the subkeywords `free_energy` , `trans_free_energy` , as detailed below.

`vibrations` **sub-tag: `free_energy`** (control.in)

Usage: `vibrations free_energy Tstart Tend Tpoints`

Purpose: governs the calculation of the free energies of vibration and rotation (rigid rotor approximation)

The calculation of the free energy of vibration and rotation (rigid rotor approximation) can be requested with this keyword. They are calculated between the temperatures `Tstart` (K) and `Tend` (K) with a total of `Tpoints` steps. The implemented equations read:

$$F_{\text{vib}}(T) = \sum_{i}^{3N-6} \left[\frac{\hbar\omega_i}{2} + k_B T \ln\left(1 - e^{\frac{-\hbar\omega_i}{k_B T}}\right) \right] \quad , \tag{4.5}$$

with N being the number of atoms in the molecule and ω_i depicting the vibrational frequencies.

$$F_{\text{rot}}(T) = -\frac{3}{2} k_B T \ln \left[\frac{2k_B T}{\hbar^2} (I_A I_B I_C)^{1/3} \pi^{1/3} \right] \quad , \tag{4.6}$$

where I_A, I_B, and I_C depict the moments of inertia of the molecule.

`vibrations` **sub-tag: `trans_free_energy`** (control.in)

Usage: `vibrations trans_free_energy pstart pend ppoints`

Purpose: governs the calculation of the translational free energy

The calculation of the free energy of translation can be requested with this keyword.

It only works if `vibrations free_energy` is used as well. The free energy of translation is calculated within the temperature range given with `vibrations free_energy` and between pressures `pstart` (Pa) and `pend` (Pa) with `ppoints` steps. The equation reads:

$$F_{\text{trans}} = -k_B T \left[\ln \frac{k_B T}{p} + 1 + \ln\left(\frac{m k_B T}{2\pi\hbar^2}\right)^{3/2} \right] \quad , \tag{4.7}$$

with p denoting the partial pressure.

4.6.2 Python script: get_vibrations.py (non-periodic and periodic (Γ-point only) systems)

A newer implementation of the same technique (finite-difference approach) as the Perl script can be achieved through a python script `get_vibrations.py` found in folder `Utilities`, and through which the running mechanism of this script is described below. The script works for nonperiodic and periodic systems (only Γ-point) as well.
`python get_vibrations.py [options] <name> <mode>`
The script functions in three modes:

- mode 0: The script creates folders containing the aims input files for single calculations. These are the 6N calculations for the N unconstrained atoms in the geometry.in in the execution path. The atoms are displaced by +/-delta (default 0.0025Å) in x, y, and z. Moreover, the control.in is copied to all folders.

- mode 1: The script creates folders containing the aims input files for force calculations. These are the 6N calculations for the N unconstrained atoms in the geometry.in in the execution path. The atoms are displaced by +/-delta (default 0.0025Å) in x, y, and z. Moreover, the control.in is copied to all folders with force output flags. The hessian matrix, eigenvectors, normal modes and their corresponding frequencies are calculated and saved.

- mode 2: The script creates folders containing the aims input files for force calculations. These are the 6N calculations for the N unconstrained atoms in the geometry.in in the execution path. The atoms are displaced by +/-delta (default 0.0025Å) in x, y, and z. Moreover, the control.in is copied to all folders with polarization/dipole, DFPT or both calculations flags depends on what the user decides to calculate IR, Raman or both.

The available options with the discussed modes:

- -r : Running aims, it should be followed with the path to aims executable (mode 2 and 3).

- -s : To run more than one core (mode 2 and 3).
- -x : Relaxing the structure before proceeding with vibrational calculations (mode 1,2 and 3).
- -d : The amount of displacement in AA along Cartesian coordinates(mode 1,2 and 3), default is 0.0025 AA.
- -p : Plotting the spectra (Raman/IR) (mode 2 only).
- -b : Broadening for IR and Raman spectrum in cm^{-1}, default is 5.
- -w : If imaginary frequencies are present, then files of the system displaced along the geometries of the imaginary frequencies are outputted (mode 2 and 3).
- -m : Printing the geometries, frequencies and vibrations in molden format.
- -M : For calculating both Raman and infrared intensities (mode 2 only).
- -R : For calculating Raman intensities only (mode 2 only).
- -I : For calculating infrared intensities only (mode 2 only).
- -n : For calculating infrared intensities of *periodic* systems the polarization grid should be specified (-n nx ny nz), **should be added after -M or -I**.
- --kx --ky --kz : Each should be followed by three integers, these options represent the polarization grid along reciprocal lattice vectors 1,2 and 3 respectively (**should be added after -M or -I**).

For further details about the calculation of the polarization of solids from the Berry phase formalism, please see Section 3.32, where the k-point grid mentioned above is better explained.

Examples

1. python get_vibrations.py name 0 -x : Creating all folders as discussed above including an additional file for relaxation.
2. python get_vibrations.py name -r $path - to - aims - excutable$ 1 -w: Creating all folders as discussed above, aims will be running in these folders one after the other. The output files will be:

- masses.name.dat : Atomic mass of atoms in a.m.u and their position as in geometry.in .
- hessian.name.dat : Hessian matrix before mass weighting.
- massweighted_Hessian.name.dat : Mass weighted hessian matrix (Dynamical matrix.)

- mass_vec.name.dat : Diagonal matrix of $1/\text{sqrt(mass)}$
- eigen_vectors.name.dat: Normal modes written in a matrix form, each column corresponds to a normal mode arranged from lower to higher frequency.
- car_eig_vec.name.dat : Eigen vectors that are not normalized (not mass weighted).
- normalmodes.name.dat: Normal modes and their corresponding frequencies.
- name.distorted.vibration : Geometry displaced along imaginary frequency mode (if exists).

3. python get_vibrations.py name -r $path-to-aims-excutable$ 2 -R -p -m $>$ output:
Creating all folders as discussed above including an additional with no previous relaxation, aims will be running in these folders one after the other. The output files will be:

- masses.name.dat: Atomic mass of atoms in a.m.u and their position as in geometry.in .
- hessian.name.dat: Hessian matrix before mass weighting.
- grad_dipole.name.dat: Derivative of dipole moment with respect to displacement delta.
- name.Raman: Raman intensities and the corresponding frequencies.
- name.molden: Molden geometries and displacements for the mode.
- name.xyz: All vibrational displacement, this file can be opened using jmol to view the vibrations.
- name_Raman_spectrum.pdf: Raman spectra plot

4. python get_vibrations.py name -r $path-to-aims-excutable$ 2 -I $--$kx 4 2 2 $--$ky 2 4 2 $--$kz 2 2 4 -p $>$ output:
Creating all folders as discussed above including an additional with no previous relaxation, aims will be running in these folders one after the other to find the infrared intensity for periodic system. The output files will be:

- masses.name.dat: Atomic mass of atoms in a.m.u and their position as in geometry.in .
- hessian.name.dat: Hessian matrix before mass weighting.
- grad_dipole.name.dat: Derivative of dipole moment with respect to displacement delta.
- name.ir: Infrared intensities and the corresponding frequencies.

- name.molden: Molden geometries and displacements for the mode.
- name.xyz: All vibrational displacement, this file can be opened using jmol to view the vibrations.
- name_IR_spectrum.pdf: IR spectra plot

It is important to note that the atoms that will not enter in the vibrational analysis are followed with constrain_relaxation .true. in geometry.in file

Theory

The theory behind the calculation of the hessian, harmonic Raman and infrared intensity is explained in the previous subsection of the **Perl script**. However, for the IR calculation in the nonperiodic system, the infrared is calculated directly from the dipole, while, in periodic systems it is calculated from the polarization multiplied with the unit cell volume.

4.6.3 Vibrations and Polarizability by DFPT within FHI-aims (non-periodic systems)

The following lists the keywords to activate the calculation of the vibrational frequencies with DFPT.

Tag: `DFPT vibration` (control.in)

Usage: `DFPT vibration [subkeywords and their options]`

Purpose: Allows to calculate vibrations using density-functional perturbation theory, use Acoustic Sum Rule (ASR) to get Hessian matrix, do not use moving-grid-effect.

Usage: `DFPT vibration_with_moving_grid_effect [subkeywords and their options]`

Purpose: give the results for vibrations with moving-grid-effect, do not use ASR for Hessian matrix.

Usage: `DFPT vibration_without_moving_grid_effect [subkeywords and their options]`

Purpose: give the results for vibrations without moving-grid-effect, ONLY served as comparison with vibration_with_moving_grid_effect.

Tag: `DFPT vibration_reduce_memory` (control.in)

Usage: `DFPT vibration_reduce_memory [subkeywords and their options]`

Purpose: Allows to calculate vibrations density-functional perturbation theory by using nearly the same memory as DFT. At present, functionals LDA, PBE are supported, relativistic is also supported. It should be noted that PBE and PBE+TS is supported only for DFPT cycle (first-order-H), but not for Hessian. Only linear-mix (no Pulay-mixer) can be used for DFPT vibration_reduce_memory at present.

Here is an example, the following need to be added to control.in:

```
DFPT vibration_reduce_memory
DFPT_mixing   0.2       #default is 0.2
DFPT_sc_accuracy_dm   1E-6  # default is 1.0d-6
```

Tag: `DFPT polarizability` (control.in)

Usage: `DFPT polarizability [subkeywords and their options]`

Purpose: Allows to calculate polarizability for cluster systems using density-functional perturbation theory.
For "DFPT polarizability", functionals LDA, PBE, HF(RI-V) are supported, relativistic is also supported.

Here is an example for using DFPT polarizability, the following need to be added to control.in:

```
DFPT polarizability
DFPT_mixing   0.5             #default is 0.2
DFPT_sc_accuracy_dm   1.0d-6  # default is 1.0d-3
dfpt_pulay_steps 6            # default is 8
```

4.6.4 Phonons via FHI-vibes and Phonopy (periodic systems)

The recommended way to compute phonon properties with *FHI-aims* is through the Python package *FHI-vibes* [132]. *FHI-vibes* is a python package built on top of the Atomic Simulation Environment (ASE, [146]), and tightly integrates *FHI-aims* with *phonopy* [227]. With that, *FHI-vibes* provides a unified user interface to run and process the necessary calculations. It thus allows harmonic phonon calculations via *phonopy* [227], but also gives access to more advanced analysis and post-processing methods, especially for the study of anharmonic effects.

Details on how to obtain, set up, and use *FHI-vibes* –including tutorials on how to perform phonon calculations– can be found in the official documentation: https://vibes.fhi-berlin.mpg.de/.

Alternatively, it is also possible to use *phonopy* directly (without FHI-vibes) by using the interface to *FHI-aims* included in *phonopy* (Version 2.5 and higher). Documentation and examples can be found in the phonopy source code at https://github.com/phonopy/phonopy/tree/develop/example/diamond-FHI-aims (cf. README.md).

Please note: The former interface *phonopy-FHI-aims* is not officially supported anymore. The last *FHI-aims* release including *phonopy-FHI-aims* and respective documentation was the 210226 release.

Note that the former, native phonon infrastucture in FHI-aims is no longer needed. The previous make targets `phonons` and `phonons.mpi` still exist, but we no longer recommend them. FHI-vibes/Phonopy is, simply, more stable and general.

4.6.5 Phonons by DFPT within FHI-aims (periodic systems)

A third level of infrastructure for vibrations and phonons in non-periodic and periodic systems based on density functional perturbation theory (DFPT) is nearing readiness and usable to some degree. However, parts of this implementation are not yet optimized. The DFPT implementation should therefore still be considered experimental at this time. We recommend to cross-check the DFPT results always with finite-difference methods described above.

Tag: `DFPT phonon` (control.in)

Usage: `DFPT phonon [subkeywords and their options]`

Purpose: Allows to calculate phonon (real space method) for PBC systems using density-functional perturbation theory. This method could get force constants using real space method and give the phonon band structures. At present, only functionals LDA without relativistic is supported.

Here is an example for using DFPT phonon, the following need to be added to control.in:

```
DFPT phonon
DFPT_mixing   0.5             #default is 0.2
DFPT_sc_accuracy_dm   1.0d-6  # default is 1.0d-3
dfpt_pulay_steps 6            # default is 8
```

Tag: `DFPT phonon_reduce_memory` (control.in)

Usage: DFPT phonon_reduce_memory [subkeywords and their options]

Purpose: Allows to calculate phonon (reciprocal space method) at q point for PBC systems using density-functional perturbation theory. At present, this keyword only works to get dynamic matrix at q = 0. This feature is under developing. At present, functionals LDA, PBE are supported, relativistic is also supported. It should be noted that PBE and PBE+TS is supported only for DFPT cycle (first-order-H), but not for Hessian.

Here is an example for using DFPT phonon_reduce_memory, the following need to be added to control.in:

```
DFPT phonon_reduce_memory
DFPT_mixing   0.5             #default is 0.2
DFPT_sc_accuracy_dm   1.0d-6  # default is 1.0d-3
dfpt_pulay_steps 6            # default is 8
```

4.7 Restarting FHI-aims calculations

This section gives background information on restarting FHI-aims calculations from the electronic structure.

Note that this section covers two versions of the restart process:

- The `elsi_restart` keyword, which is simpler and somewhat more flexible in case the objective is just to restart a given electronic structure calculation from the previous one
- The `restart` keyword, which produces differently formatted output and which supports a few more sophisticated tasks, for example, restarting using a rotated version of a structure calculated earlier.

If you just want to restart an electronic structure calculation in a flexible way, we suggest use the `elsi_restart` keyword. `elsi_restart` produces the same file format, irrespective of whether the linear algebra is performed in serial or in parallel and irrespective of how many MPI tasks were used. See the definition of that keyword for a brief explanation and for a few more bells and whistles (e.g., some form of density matrix extrapolation along the way, if the geometry is slightly adjusted prior to the restart).

Both `elsi_restart` and `restart` are regression tested at the time of writing.

4.7.1 Restart procedure based on the "elsi_restart" keyword

This is the most flexible restart procedure in FHI-aims at the time of writing, if the objective is simply to restart a calculation. It writes a compressed version of the density matrix or, in periodic calculations, a compressed version of the density matrix at each $k-$point. (This means that many separate restart files can emerge for dense k-point grids.)

To read or write a restart file or set of restart files in every scf iteration, simply insert the following line into your `control.in` file:

```
elsi_restart read_and_write 1
```

This command will prompt FHI-aims to read its density matrix input files if they exist in the directory in which the code is executed. If they do not exist, the code will proceed assuming that this is a calculation from scratch. However, in either case the code will write new restart files in every single s.c.f. iteration and it will overwrite earlier restart files.

To have restart information written in (say) only every tenth s.c.f. iteration, just modify as follows:

```
elsi_restart read_and_write 10
```

To only write restart files (but not read), use

```
elsi_restart write 10
```

(of course, the number "10," which defines the number of s.c.f. iteration after which a new file will be written, can be customized).

Conversely, to only read existing files but not write new ones, use this version:

```
elsi_restart read
```

Finally, it is possible to write restart files for one density functional and then use them to restart the s.c.f. cycle with another density functional. For example, you might wish to preconverge the electronic structure of a given system using a computationally simple functional first (e.g., PBE) and then restart from that density matrix but using a more costly functional, e.g., a hybrid functional.

This process is supported but it is important to note that by default, FHI-aims works on a reduced (time-reversal symmetric) k-space grid for scalar relativistic semilocal DFT. In contrast, hybrid DFT requires all k-points to be present. To include all k-points in the semilocal DFT calculation as well and write all necessary restart files, please add the keyword `symmetry_reduced_k_grid .false.` to `control.in`:

```
symmetry_reduced_k_grid .false.
```

Again, for more information please go to the `elsi_restart` keyword definition.

4.7.2 Restart procedure based on the "restart" keyword

Calculations can be restarted from previous wavefunctions using the keyword `restart`. In general, restarting the same calculation on the same computer using the same number of MPI-tasks will always work.

Important — Rotated systems: FHI-aims wavefunctions are not symmetry-aware. While the same restart file can be used for molecules *translated* by a vector, this does not work for *rotated* molecules. For a way to restart systems with rotated geometries or superpositions of molecular densities see Section 4.7.5.

FHI-aims has two main means of saving the wavefunction for a later restart, depending on the used keywords.

Kohn–Sham eigenvector (*"wavefunction"*): This is the default case always used for calculations using `KS_method serial`. The saved file contains some information on the system (number of basis functions, states, spin, k-points), the Kohn–Sham eigenvector, eigenvalues and occupations.

Kohn–Sham density matrix (*"density matrix"*): This variant is only used for calculations using `KS_method parallel`. The saved file contains the density matrix ($\hat{n}_{ij} = \sum_l f_l c_{il} c_{jl}$), either in full or sparse storage.

The corresponding keywords determining which version of storage is used are `KS_method` and `density_update_method`. See Fig. 4.10 for possible options and outcomes.

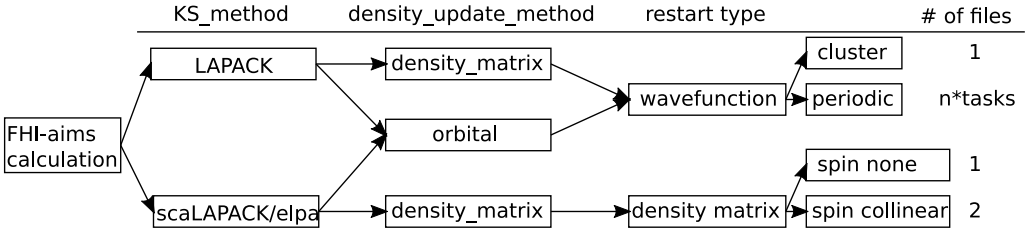


Figure 4.10: Scheme of different restart types depending on used methods.

For technical reasons there is not one single way, but the above mentioned variants with again, variants.

The *density matrix* based restart always writes one file per spin-channel, no matter how many parallel MPI tasks are used. The *wavefunction* based restart generally writes a single restart file for all cluster calculations and one file per MPI-task for periodic calculations. Due to this periodic calculations using the *wavefunction* restart need to be restarted with the same number of MPI-tasks.

4.7.3 Mixing variants - the "force_single_restartfile" option

For a subset of possible calculations the above described differences are not valid and it is possible to restart different calculations using one type of restart file. This is the case for all possible cluster calculations (using either `KS_method serial` or `KS_method parallel`) and for periodic calculations with only 1 k-point, Γ-only (using either `KS_method serial` or `KS_method parallel`.

This is implemented via the keyword `force_single_restartfile`. Using this functionality, it is for example possible to profit from the parallel ELPA solver for huge systems and still get a file containing the KS-wavefunction for post-processing.

4.7.4 Comments on the 'restart' starting point and on self-consistency

As noted in the description of the `restart` keyword, it is important to note that the `restart` infrastructure corresponds to a restart from the last Kohn-Sham orbitals, not from the last density.

In practice, this means that the code will restart from the last *unmixed* Kohn-Sham density, not from the last *mixed* density. When restarting from a non-selfconsistent starting point, this can lead to unexpected jumps in the calculated non-selfconsistent total energy between the "old" and the "new" (restarted) FHI-aims run.

Only the self-consistent total energy is truly meaningful. This quantity (the self-consistent) total energy should be the same for the same stationary density, when approached from different starting densities. However, note additionally that some systems may exhibit several different self-consistent stationary densities even for the exact same atomic positions and for the exact same density functional. A simple example are antiferromagnetic vs. ferromagnetic spin states in some systems. In such cases, the true ground state in a DFT sense is the stationary density that yields the lowest energy. It can be found by way of a global search for different stationary densities, usually by varying the initial density guess.

4.7.5 Rotating the FHI-aims wavefunction

This feature is new and tested for the case of weakly interacting organic molecules as can be found in organic crystals. If you encounter any problems or strange behaviour, please let us know!

In many molecular systems a rotation does not change the electron density (or wavefunction) of a system. Therefore, the same wavefunction can be used again to restart such a calculaction.

This is especially true for weakly interacting systems such as van-der-Waals bound organic crystals. In these cases, the wavefunction of each single molecule in the crystals is not too different from the isolated wavefunction of a single molecule. This can be exploited to start a FHI-aims calculation of huge molecular systems from a *superposition of molecular densities* instead of the usual *superposition of atomic densities*.

This functionality is implemented as an external tool using Python and can be found in the FHI-aims repository itself (Python-package *aimsutils*) or at `https://gitlab.lrz.de/theochem/aimsutils`.

If you use this functionality, please cite *C. Schober, K. Reuter, H. Oberhofer, J. Chem. Phys., 144:054103, 2016*[209].

Important pre-requisites

This functionality only works for restart files containing the wavefunction of the system, not the density matrix. Therefore, please make sure you have wavefunction restart files or use the option `force_single_restartfile` . The FHI-aims output file of the calculation is also necessary.

Rotations can be defined via Euler angles in *zyz* convention or quaternions. To avoid any confusions with rotational conventions (especially with Euler angles), the input geometry will be rotated and saved using the same rotation matrix. Please be sure to check if your expected and the actual rotation are the same (or directly use the generated `geometry.in`).

The final rotated restart file is again in the wavefunction format.

- `example_calc`

```
|
+-- control.in
+-- geometry.in              # necessary
+-- aims.out                 # necessary
+-- restart.wavefunction     # name can be anything, will be parsed from outfile
```

Examples

The two examples are shown using pre-defined scripts, but it is totally possibly to use the Python package directly. The full Python API is available via the Sphinx-documentation of the *aimsutils* package.

Rotation of a single system: The script `RotateSingle.py` can be used to create the rotated geometry and restart for a single calculation.

```
$: # RotateSingle alpha beta gamma -t x y z
                   '-----.-----'    '---.---'
                   Euler angles   translation vector
$: RotateSingle 45 0 123 5.4 2.3 6.3

$: # RotateSingle x_i x_j x_k x_l -t x y z
                   '------.------' '---.---'
                     quaternion   translation vector
$: RotateSingle 0.33 -0.12 0.44 0.84 5.4 2.3 6.3
```

This will create a subfolder *rotated* with the new `restart.combined` and `geometry.in`.

Rotation and combination of wavefunctions The script `RotateMany.py` reads a instruction file (`rotations.in`) with the following format:

```
# Comments or empty lines are ignored

# title will also be foldername
title project_x
# lattice vector for new cell if any
lattice_vector 11.2 0.00 0.00
lattice_vector 0.00 10.5 0.00
lattice_vector 0.00 0.00 10.7
# now all rotated molecules
# Euler angles (zyz)
# alpha beta gamma x y z parent_molecule
33.4 12.4 167.0   10.  0.  0.  calc_2
 0.  45.    0.     3.5 2.1 5.4 calc_1
130.  90.   23.     0.  2.6 12. calc_3
# OR Quaternion
```

```
# x_i x_j x_k  x_l   x y z
-0.4 0.15 0.33 -0.84 4 0 0 calc_2
-0.2 0.45 0.13  0.84 0 2 0 calc_1
 0.0 0.00 0.00 -0.32 4 4 5 calc_3
```

The necessary converged FHI-aims calculations must be in subfolders with the appropriate name:

```
- calc_something/ (main calculation folder)
|
+-- rotations.in
+-- calc1/
    |
    +-- ...
+-- calc2/
    |
    +-- ...
+-- calc3/
    |
    +-- ...
```

This will create a subfolder `project_x` with the rotated wavefunction assembled from the individual calculations defined in the script (`restart.combined`) and geometry (`geometry.in`).

Theory

The following should give a summary of how the rotation of wavefunctions can be done with FHI-aims. For details on the used methods and mathematics, please have a look at the references.

In FHIaims, a basis function is defined as

$$\Phi_{i,lm} = \frac{u_i(r)}{r} \cdot S_{l,m}(\theta, \phi), \tag{4.8}$$

with $S_{l,m}(\theta, \phi)$ being real valued spherical harmonics. These are obtained from the complex spherical harmonics Y_{lm} via

$$S_{l,m}(\theta, \phi) = \begin{cases} \frac{(-1)^m}{\sqrt{(2)}}(Y_{lm} + Y^*_{lm}) & m > 0 \\ Y_{l0} & m = 0 \\ \frac{(-1)^m}{i\sqrt{(2)}}(Y_{l|m|} - Y^*_{l|m|}) & m < 0 \end{cases} \tag{4.9}$$

A wavefunction is then given by

$$\Psi_k(r) = \sum_{i=1}^{\text{n\_basis}} c_i^k \Phi_i(r) \tag{4.10}$$

with the coefficients c_i^k.

A rotation of the molecule (rotation matrix $\mathbf{R}$ leads to the same set of YLMs $\mathcal{Y}$ (as they are fixed with respect to the xyz-coordinate system), but with different coefficients $\mathbf{c}$ (new linear combination of basis functions):

$$\mathbf{c}' = \mathbf{R}\mathbf{c} \tag{4.11}$$

The rotation of complex YLMs can be done with Wigner D matrices ($D^l_{mm'}$). For real YLMs different schemes are available[153, 11, 26]. Due to the non-default YLM convention in FHIaims we construct our rotation matrices for different l from the complex Wigner D matrices via the transformation matrix $\mathbf{C}^l$ (reference: [26]):

$$S_{l,m} = \mathbf{C^l} Y_{l,m} \tag{4.12}$$

The matrix $\mathbf{C}$ is constructed according to [26] with the constraint of the different sign convention in FHIaims. The real rotation matrix $\Delta^l(R)$,

$$\Delta^l(R) = \left(\mathbf{C}^l\right)^* \mathbf{D}^l(R) \left(\mathbf{C}^l\right)^t, \tag{4.13}$$

is then used to obtain the rotated coefficients for each l, m for each basis function in the system.

$$\mathbf{c}'^l = \Delta^l(R)\mathbf{c}^l \tag{4.14}$$

4.8 Finding Transition States: the aimsChain

This project aims to provide an all-in-one package for various flavours of the chain of states methods for finding the minimum energy path(MEP). Currently the nudged elastic band method (NEB)[90, 101], the string method[237], and the growing string method[188] are included.

The package was maintained by *yaoyingyu@hotmail.com*, who, in the past, had offered to help with issues related to the aimschain code. At the time of writing (2021), the aimschain works, but is effectively maintained collectively by the FHI-aims community. Please use our regular community channels to contact us for assistance. More importantly, please consider contributing fixes and updates to the code and to its documentations if you encounter any issues.

The aimsChain code is distributed under the Lesser General Public License as published in Version 3, 29 June 2007, at http://www.gnu.org/licenses/lgpl.html . Some of the optimizer routines in the code originated within the Atomic Simulation Environment (ASE). We wish to give full credit to the developers of these routines. The aimsChain code can also be found in a separate distribution that was maintained by Yingyu Yao. The reason we distribute them directly within the FHI-aims repository is for the convenience of all FHI-aims users, but again: We wish to give full credit to the work and the license of the original authors within ASE.

4.8.1 Installation

The package is located in `utilities/aimsChain`.

Importantly, the aimschain is currently written for python2. Thus, you will need a working python2 installation to use it. Such an installation is usually provided by the system administrators on the computer you are using (in some cases, this administrator might be yourself).

In general, any computer/cluster that is capable of running FHI-ams and has python2 numpy/scipy support can be used.

To install, user should follow these steps:

1. Copy the package to the directory where you normally install software, e.g. `~/local/bin` or your home directory. This path will be referred to as <package path> below.

2. Modify your shell environment to make it aware of the python2 environment and of the relevant python software that is needed to run aimsChain. For instance, on some computers this tsk could be achieved by modifying your `~/.bashrc` to include lines similar to the following ones:

```
export PATH=$PATH:<package path>/tools
export PYTHONPATH=$PYTHONPATH:<package path>
module load <necessary modules>
```

These modifications are recommended but not necessary. You can also add them to your job script instead if you do not want to pollute your `.bashrc`. Note that the necessary modules – or equivalent modifications – to set up command line variables correctly for python2 and for scipy will be specific to your computer and you will need to make sure to use and/or install the necessary software on your computer.

The scipy and numpy packages depend on some other packages that need to be present on a computer and they need to be findable from your command line. For example, a version of the "basic linear algebra subroutines" that is compatible with the available numpy package must be present and findable via the "LD_LIBRARY_PATH" environment variable.

If you managed to set up python2 / numpy / scipy and the software on which they depend on your computer, you are good to go.

3. You are done.

To test your installation, source the `.bashrc` file and start a python shell. The package is successfully linked if executing

```
import aimsChain
import scipy
import numpy
```

reports no error. Otherwise please double check the software installed on your computer and resolve any installation problems.

4.8.2 A Quick Start

While offering an extensive list of keywords, the package is aimed to work with most systems out of the box. Often very little configuration is required to conduct a successful path search. The keywords are available, on the other hand, to make the package as adjustable as possible, so that it can be finely tuned to tackle the less conventional systems.

In the package directory, a few sample jobs have been prepared, imitating several different possible use cases. Please modify `chain.in` in each sample (other than sample 1) so that `run_aims` links to your own aims executable.

- sample 1: This is a graphical example on a 2D analytic surface that demonstrates the principles behind the chain-of-states methods. You can skip it if you have used NEB/SM elsewhere.

  You can and should run this demo in your personal computer instead of the cluster, since it is not actually calling FHI-aims for calculations.

  To start, you should go to `samples/sample1/run` and execute `job.sh`. This will execute the MEP finding algorithm on the 2D LEPS potential energy surface.

To see the resulting process, proceed to `samples/sample1` and run `gnuplot plot.gnu` to see the evolving path on the surface.

This is also the perfect test bench for testing with different configurations. However keep in mind that performance in 2D is very different from the 3N-D space in reality, and therefore this example should not be used for performance optimization.

- sample 2: The inversion of an ammonia molecule. This example surveys the MEP with 8 images, using an very tight convergence setting(`force_thres` =0.01), which is too tight for most of the more complex systems, and will require tremendous amount of computation for non trivial systems in general. The resulting path would represent a very accurate representation of the MEP.

- sample 3: This example measures the energy barrier to transfer a methyl group from one chlorine atom to another. It is using 6 images, coupled with a relatively loose convergence setting(`force_thres` =0.2). The resulting MEP from this is in general not accurate enough for a quantitative analysis. However, climbing image is turned on with a relatively tight convergence setting(`climb_thres` =0.05). The result is an accurate energy barrier calculation.

  Note that in this example we are also utilizing `constrain_relaxation` in our input geometries to limit the movement of atoms. In this particular case, one of the chlorine atom is fixed at the origin, and the others are constrained to move along the x-axis.

- sample 4: In this example we demonstrate how to start from an external set of initialization images . We are calculating the MEP for a non-trivial isomerization. Geometries from an previous calculation is provided as the starting geometries.

- sample 5: This sample provides an insight into MEP searching with periodic systems. It involves only the `interpchain.py` provided in `aimsChain/tools`, which should be already in your $PATH if you followed the installation process. This tool will generate an initial path exactly the same way as the actual job script does. The result is printed into a directory named `interpolation`. A multi-frame *.xyz* file is written, as well as one *.in* geometry file for each image.

  Sample 5.1 and 5.2 are the same geometry, the only difference is that periodic interpolation is set to `true` in the former, and `false` in the latter. Run `interpchain.py` in each directory and inspect the resulting path with your favourite visualization software. In sample 5.1 we acquire a smooth interpolation, but in sample 5.2 with periodic interpolation turned off, one row of atoms had to travel across the entire cluster.

  Another way to do a correct interpolation is shown in sample 5.3. You can manually adjust the coordinate of the atoms so that they are set at the correct coordinate with respect to the initial geometry. `fin.in` is adjusted in this case to reflect the change. This way, the interpolated path is again correct, even though `periodic_interpolation` is turned off. This is the preferred method of dealing with periodic systems, because the automatic interpolation finds the shortest path, but not necessarily the correct path. When working with periodic systems

you should always do a `interpchain.py` first and inspect the result before running the actual job.

- sample 6: Here we show when and why the growing string(GSM) method should be used. Here the isomerization process involves rotation of in dihedral angles, and linear interpolations would result in overlapping atoms, which can be checked using the `interpchain` tool.

  The traditional approach dealing with these cases is to rotate the atoms manually and generate a non-overlapping initial path. GSM provides an alternative to this approach by growing the string from end points, which eliminates the need for manually generated initial path.

- sample hydra: It is essentially the same as sample 1, but prepared for the hydra cluster. This demonstrates the required modifications when transferring between different platforms.

The samples include the following files.

- ini.in: initial geometry in the standard aims format
- fin.in: final geometry in the standard aims format
- control.in: standard control file for aims. This control will be used for all aims calculations.
- job.sge: job submission script, which you submit into the cluster.
- chain.in: control file for aimsChain, see below for a detailed description

If everything is set up properly, you can simply `qsub job.sge` to get the job running. When running on clusters not using SGE, the job scripts must be changed accordingly.

4.8.3 Configuration

aimsChain control file

The chain control file is the file that governs the evaluation of MEP. It follows the same configuration convention set by the aims `control.in`, namely a list of key value pairs. The control file should be named `chain.in`.

This is a sample control file.

```
#A sample chain.in
run_aims mpiexec -ppn 8 -n $NSLOTS ~/bin/aims.062812.scalapack.mpi.x
initial_file ini.in
final_file fin.in
n_images 6
force_thres 0.2
```

```
use_climb true
climb_thres 0.05
```

All the available keywords for the control are listed below:

Tag: `run_aims` (chain.in)

Usage: `run_aims` your command
Purpose: Provide the command that is used to call aims in your environment.
Default value: mpiexec -ppn 8 -n $NSLOTS ~/bin/aims.081912.scalapack.mpi.x

You will need to change the default value for whatever command you are using.

There is no need to quote your command, simply type it as you would in a bash shell. Everything after the keyword are concatenated into a single string.

Tag: `initial_file` (chain.in)

Usage: `initial_file` filename
Purpose: Provide the initial geometry file for the path.
Default value: ini.in

The format should be the same as the aims `geometry.in`. Flags such as constraints are interpreted, and will affect the evaluation of MEP. All the intermediate images will have the same constraints as the initial geometry.

Atoms in the initial and final geometry must establish a one to one correspondence. The ith line in the initial geometry and the final geometry must correspond to the same atom. Often there are many possible combinations, it is the user's responsibility to use the most physical one and/or check different combinations. You can perform a test interpolation with the `interpchain.py` tool provided.

Tag: `final_file` (chain.in)

Usage: `final_file` filename
Purpose: The final geometry for the path. Refer to `initial_file` .
Default value: fin.in

Tag: `n_images` (chain.in)

Usage: `n_images` value
Purpose: The number of images used for MEP calculation.
Default value: 5

Suitable value for this variable is highly dependent on the desired degree of resolution and the complexity of the system. A simple single barrier reaction will require 5 images to achieve a good result, while more complex potential energy surfaces require roughly 3 or more images per hill/trough.

If the geometry is not computationally expensive(tens of seconds per calculation with a reasonable # of CPUs), the simple principle of "the more the merrier" can always be applied.

Tag: `external_geometry` (chain.in)

Usage: `external_geometry` file_name
Purpose: If this variable is set, the initial set of images will be obtained from a external source instead of a linear interpolation.
Default value: None

This flag should be set to the filename of a text file listing all of the intermediate geometries in order, e.g.

```
./geo/1.in
./geo/2.in
./geo/3.in
```

It is advised that if you have anything better than a direct linear interpolation, you should use it as the initial guess. Linear interpolation can be highly inefficient, and would result in cases which never converges. When in doubt, you can always use the growing string method.

Tag: `periodic_interpolation` (chain.in)

Usage: `periodic_interpolation` true/false
Purpose: Whether or not interpolation is done while considering periodic boundary.
Default value: False

This setting affects periodic systems only. If set to true, each atom's final geometry will be checked against all periodic counterparts to find the shortest path between the initial and final geometry. This can resolve some problems caused by the periodic boundary condition. Consult sample 4 for detailed usage. Always use the provided `interpchain.py` tool to check the resulting interpolation and confirm that it's physical before running.

Tag: `resample` (chain.in)

Usage: `resample` true/false
Purpose: Resample the input path into arbitrary number of images.
Default value: False

This tag is effective only when `external_geometry` is set. When `resample` is set to true, the package will resample the path you provided and interpolate it to have # of images equal to `n_images` . This can be very useful if you find your calculation to have an inadequate number of images. You can extract the geometries from the final iteration, and start a new job by resampling them.

Tag: `aims_restart` (chain.in)

Usage: `aims_restart` value
Purpose: Reuse the wave function restart file to speed up the calculation. This should set to be the same as `restart` in `control.in`.
WARNING! This function is not compatible with aims as of July 2013, and only works occasionally.
Default value: None

Value should be the same as `restart` in your `control.in`. The wave function restart file will be copied from iteration to the next, so that the scf cycles can start from an existing configuration. When working correctly, this can cut the computation time required by half or more.

However, at this stage `restart` has very limited check on the input restart file, resulting in an error when the wave functions don't match. Do not use this keyword unless you are sure that your version of aims will start a new scf initialization instead of reporting errors when the wrong restart file is provided. If certain of which, however, the keyword should always be used to speed up the calculation.

Setting `restart` will consume a decent amount of disk space. You may want to remove the restart files if you are planning to store the calculation permanently.

Tag: `restart` (chain.in)

Usage: `restart` true/false
Purpose: Whether the package will allow restarts. When set to true, you can simply submit the job again to continue from the previous calculations.
Default value: true

When set to true, a restart file will be written as the path iterates. When the job is submitted/resubmitted, it will first check for the restart file to see if it's possible to start from a previous calculation. This is useful, when, for example, your job is killed due to time limitations on the cluster. Please refrain from changing settings between restarts, which is always error prone. If you are planning to change some settings, it's always safer to extract the most recent geometries and use them as the initial path for a new calculation.

There is still the risk that the job is killed while the restart file is been written, in which case anything could happen when the job is resumed. However, considering the rarity of such event, it should not be a problem in practice.

Tag: `method` (chain.in)

Usage: `method` neb/string
Purpose: Pick the method to use for the MEP calculation.
Default value: string

Currently only string method and NEB are supported. In our testing, string method have shown to be slightly more stable than NEB, and therefore is our recommended method.

Tag: `neb_spring_constant` (chain.in)

Usage: `neb_spring_constant` value
Purpose: This sets the spring constant used in NEB calculation, and has no effect if `method` is string.
Default value: 20.0

It should be set to have the same magnitude as the force felt by the geometry, which is hard to determine before hand. However a generic value of 20 is good in general. Please try `method` string first if you suspect that the spring constant have to be changed for the system to converge.

Tag: `force_thres` (chain.in)

Usage: `force_thres` value
Purpose: The threshold for convergence.
Default value: 0.2

Optimization will stop when the residual forces in the system is smaller than this preset value. If `use_climb` is true, climbing-image calculation will start after this threshold has reached. $0.2eV$ is always a good starting point, and for more rigorous calculations $0.1eV$ or $0.05eV$ can be used. Do not set it too small, since some numerical noises in the forces will always exist even when the path is well converged.

Also note that adequate value is dependent on the system. A reaction with a $3eV$ barrier can be much more well converged at $0.05eV$ threshold than a reaction with a $0.3eV$ barrier at the same threshold, because the former, very often, has a steeper barrier and hence larger force by nature.

Tag: `optimizer` (chain.in)

Usage: `optimizer` dampedBFGS/BFGS/LBFGS/trm/CG/FIRE
Purpose: This picks the optimizer used for optimization.
Default value: dampedBFGS

BFGS is a textbook implementation of BFGS optimizer, which was observed to have wild behaviour is some situations. The fact that BFGS series of optimizer can be used for this type of calculations is in fact purely coincidental. The series of force projections involved in the chain of states methods can severely hamper the effectiveness of quasi-Newton optimizers. Please resort to FIRE optimizer whenever you observe wild behaviour in the optimization process.

dampedBFGS damps the original BFGS optimizer using several techniques, and is slightly more stable in those situations. (But they still get stuck from time to time!) This is the default setting that should always be tried first.

LBFGS is the limited memory version of BFGS, which uses less memory for extremely large systems. However, given that in general FHI-aims is applied to systems with hundreds of atoms at most, this optimizer has little advantage in this front. However, due the fact that LBFGS uses only recent iterations for approximation, the results can be quite different from BFGS in a some PES. Therefore it's worth trying if BFGS fails.

trm is the same trust-radius method ported from FHI-aims. In our tests it has proven to be reasonably stable and efficient. We would recommend this as another alternative along with FIRE when dampedBFGS fails.

CG is the textbook implementation of conjugate-gradient algorithm using finite difference scheme. In our tests it has not shown any advantage over other algorithms, ans is provided for the sake of completeness.

FIRE is the Fast Inertial Relaxation Engine, one of the better non-Newton type optimizers. It is slower than BFGS series of optimizers in general, but is immune to the aforementioned instability because it does not approximate the Hessian. If dampedBFGS fails, try FIRE with `global_optimizer` off as an alternative.

Tag: `global_optimizer` (chain.in)

Usage: `global_optimizer` true/false
Purpose: Whether all images are optimized as a single object, or if individual images are optimized as separated object.
Default value: true

The global version, when coupled with BFGS-series of optimizer, is often faster than the non-global optimizer. However, this combination is less stable, and can lead to wild results. The non-global optimizer coupled with FIRE seems to be the most stable one in our tests, but is much slower than the former. Non-global optimizer combined with BFGS, on the other hand, slows the calculation significantly due to overestimation, and should be avoided.

Tag: `xyz_lattice` (chain.in)

Usage: `xyz_lattice` a b c
Purpose: This key only affects the *.xyz* file written in `paths` directory. It governs how the lattice is repeated in each lattice vector. The default is a standard 2 2 1 setting, valid for most surfaces.
It has no effect on clusters.
Default value: 2 2 1

Tag: `map_unit_cell` (chain.in)

Usage: `map_unit_cell` true/false
Purpose: This key only affects the *.in* file written in `paths` directory. It determines whether atoms in these geometries are mapped back to the central unit cell.
It has no effect on clusters
Default value: false

Tag: `use_climb` (chain.in)

Usage: `use_climb` true/false
Purpose: This will turn on the climbing image[100] feature.
Default value: false

The points with highest energies will move toward a higher energy location along the path until the saddle point is reached. Please consult `climb_mode` for a detailed explanation of this process. If you are only interested in finding the energy barrier, it's possible to set `force_thres` to some larger value, such as 0.2, and set `climb_thres` smaller(e.g. 0.05) for a reasonably tight convergence. In this case decreasing `force_thres` will not affect the accuracy of climbing image, so long as the climbing image converges. Transition state finding can be sped up significantly this way by reducing the number of single point calculations required.

Tag: `climb_thres` (chain.in)

Usage: `climb_thres` value
Purpose: Set the convergence criterion for the climbing image.
Default value: same as `force_thres`

The climbing image will stop when the residual forces in the system is smaller than this value. It is normally, although not necessarily, set to a value smaller or equal to `force_thres` . In simple systems, the climbing image is capable of pushing the system done to $meV/\text{Å}^2$ range, but that's not as plausible in larger systems.

Tag: `interpolated_climb` (chain.in)

Usage: `interpolated_climb` true/false
Purpose: Determine whether the climbing image is chosen from one of the existing images or interpolated from known energies and geometries.
Default value: true

When set to true, the current energies and geometries will be fitted with a cubic spline to identify the point with highest energy. If this geometry is sufficiently far from existing nodes, then the interpolated geometry is used. If the geometry is close to one of the existing images, than that image is set to be the climbing image.

When set to false, the image with highest energy is set to be the climbing image.

Tag: `climb_mode` (chain.in)

Usage: `climb_mode` 1/2/3
Purpose: This determines the "mode" of the climbing image.
Default value: 2

Increasing mode corresponds to increasing stability and decreasing efficiency. The main difference lies in the number of images that are allowed to move during the climbing process.

climb_mode 1: Only the image with highest energy is allowed to move. This setting can manage many circumstances, provided that force_thres is small enough so that the converged path provides a stable basin.

climb_mode 2: The image with highest energy and its two neighbouring images are allowed to move. The two additional image provides an evolving tangent estimate as the central image climbs. This mode can tackle nearly all cases where climb_mode 1 fails under the same force_thres setting. It is safe, in general, to set force_thres to 0.3 if you are only interested in energy barrier.

climb_mode 3: All of the images excluding the end points are allowed to move. This is the most stable, but rarely necessary setting. The efficiency drop is highly dependent on the system and the force_thres setting. In general this should be used as a last resort when force_thres can be converged to a tight setting but climbing image on other modes fails.

Tag: `climb_global_optimizer` (chain.in)

Usage: climb_global_optimizer true/false
Purpose: The same as global_optimizer , except this keyword is dedicated for climbing image.
Default value: true

The tag has no effect when climb_mode is 1, in which case the global and non-global optimizer are equivalent. Unlike global_optimizer , BFGS+Non-global setting can be a good combination for climb_mode 2 if the default setting fails, mainly due to the well behaving local basin for a roughly converged string.

Tag: `climb_optimizer` (chain.in)

Usage: climb_optimizer dampedBFGS/BFGS/LBFGS/CG/TRM/FIRE
Purpose: The same as optimizer , but dedicated for climbing image.
Default value: fire

Tag: `climb_control` (chain.in)

Usage: climb_control file_name
Purpose: Specify a different `control.in` for the climbing image.
Default value: control.in

It's possible, for example, to converge the entire path with a light setting, and then converge the climbing image with tight setting, which consumes far less computational power.

However, there are a few things to note if this feature is utilized. First, force_thres

must be set smaller than normal, perhaps even the same value as climb_thres . Since a different control file can produce a radically different PES, you would like to be as accurate as possible when converging the string, so that the error is not amplified with a tighter setting. Secondly, it's better to use climb_mode 2 or 3 for these kind of computations, since the single image climbing mode can rarely recover from a bad starting point.

This feature is best suitable for production works where a tighter setting which will dramatically increase the time required for single point calculations.

As an alternative, you can also kill your running process and change the control.in file manually before restarting the process.

Tag: **use_gs_method** (chain.in)

Usage: use_gs_method true/false
Purpose: This controls whether the growing string method is used or not.
Default value: False

The growing string method(GSM) is explained in sample 6. It is best suitable for cases where it is certain that linear interpolations between initial and final images will not lead to a correct path. This can be caused by movements such as rotations. It is also useful when doing large-scale automated scanning, where the user does not have the time to look at geometries on a case by case basis.

When set to true, the path will start from the two end point and slowly grow inward to generate a physical path. When the path is completely grown, it will be passed onto NEB/SM for further calculations.

Beware that when direct interpolation can lead to correct paths, e.g. surface dispersion, using growing string method may reduce efficiency.

Tag: **gs_thres** (chain.in)

Usage: gs_thres true/false
Purpose: The GSM counterpart for force_thres .
Default value: force_thres *1.5

When the forces in the system goes below this preset value, a new node will be added to the path. It should not be set too small, since the purpose of GSM is to generate a physical path, not a well-converged path. The default value gives a good guideline for this key.

Tag: **gs_n_images** (chain.in)

Usage: gs_n_images value
Purpose: The GSM counterpart for n_images .
Default value: n_images

In some circumstances, you may want to specify a different number of images for the

growing stage of the calculation. This key serves this purpose. After the growing process, the path will be re-sampled to match the value of n_images

---

Tag: `gs_optimizer` (chain.in)

Usage: gs_optimizer dampedBFGS/BFGS/LBFGS/trm/CG/FIRE
Purpose: The GSM counterpart for optimizer .
Default value: trm

For GSM, dampedBFGS, trm, and FIRE are optimizers worth trying.

Tag: `gs_global_optimizer` (chain.in)

Usage: gs_global_optimizer true/false
Purpose: The GSM counterpart for global_optimizer .
Default value: false

We have not done enough testings to determine conclusively the better set of optimizers. The default provided here have performed well in our benchmarks, but feel free to try other combinations.

Tag: `lbfgs_alpha` (chain.in)

Usage: lbfgs_alpha value
Purpose: Set the curvature used to initialize the Hessian(in $eV/\text{Å}^2$) in LBFGS. In general any value that is not magnitudes off are acceptable.
Default value: 120.0

Tag: `lbfgs_memory` (chain.in)

Usage: lbfgs_memory value
Purpose: Set the number of past iteration that LBFGS is going to remember. A larger value will increase memory consumption, and a small value will decrease accuracy.
Default value: 30

Tag: `lbfgs_maxstep` (chain.in)

Usage: lbfgs_maxstep value
Purpose: The maximum step in Å that an atom can take in a single iteration. This is similar to max_atomic_move , but defaulted to a much smaller value due to the increasing complexity.
Default value: 0.04

Tag: `bfgs_alpha` (chain.in)

Usage: `bfgs_alpha` value
Purpose: Same as `lbfgs_alpha` , but used by BFGS, trm, and dampedBFGS optimizers.
Default value: 120

Tag: `bfgs_maxstep` (chain.in)

Usage: `bfgs_maxstep` value
Purpose: Same as `lbfgs_maxstep` , but used for BFGS and dampedBFGS optimizers.
Default value: 0.04

Tag: `fire_dt` (chain.in)

Usage: `fire_dt` value
Purpose: The initial time step used by the FIRE optimizer.
Default value: 0.02

It is set to a very conservative value because FIRE is intended to be used as a fall back when BFGS fails. Values up to 0.1 can be used to speed up the calculation, provided that the PES is smooth enough. Internally, the time step is dynamically adjusted, and this key only serves to initialize the value.

Tag: `fire_maxstep` (chain.in)

Usage: `fire_maxstep` value
Purpose: Same as `lbfgs_maxstep` , but used for FIRE optimizer.
Default value: 0.04

4.8.4 Preparation before running

Creating a project directory

You should create a directory for each and every aimsChain calculation you are planning to run. It should contain the following files.

```
-Project directory
|
|--chain.in
|
|--control.in
|
|--ini.in*
|
|--fin.in*
|
```

```
 |--extgeo.lst*+
 |
 |--images*+
      |
      |-image1.in*+
      |
      |-image2.in*+
      |
      |-...
*The filename can be set by the user
+The files are only necessary for starting from external geometry.
```

geometry file

The initial and final geometry should be in the standard aims input format. Keywords such as constraints will be interpreted. The `constrain_relaxation` tag should be used with caution–it acts as a double-edged sword in terms of MEP evaluation. You can try to remove the constraint tag from sample 2 and observe the efficiency boost for example. In other cases, setting it may help convergence by reducing degree of freedoms.

In addition, you should pre-align your initial and final geometries to remove any rotational/translational component in the geometry, this will reduce the computational effort required.

The ordering of atoms in the geometry file is very crucial. The atoms in the initial and final geometry must establish a one to one correspondence, so that the ith line in the initial and final geometry represents the same atom. Changing the order can change the result from the evaluation dramatically.

For example, consider the diffusion of benzene on a surface. By changing the ordering of atoms you may force the diffusion to be done via rotational or translational motion, which would result in very different MEP. If you can visualize more than one possible combination, try all of them to find the lowest barrier. It is a matter of fact that there often exists more than one MEP between two geometries, but the overall barrier is only determined by the MEP with lowest barrier. Chain of states methods will evolve toward the MEP that's closest to the initial path, but not necessarily the lowest in energy.

aims control file

The aims control file should contain two lines.

```
compute_forces .true.
final_forces_cleaned .true.
```

This will turn on the force evaluation for aims, which is required for any chain of states method. `relativistic` , when required, must be set to `atomic_zora scalar`, since we require force evaluations.

Sometimes additional configurations can require more than the two lines listed above. For example, when using `b3lyp` you need to set

```
RI_method lvl_fast
```

for a correct force evaluation. Generally you can find these information in the aims output. A rule of thumb is to relax the geometry using your configurations and skim through the aims output for any warnings. This will force aims to provide all warnings related to the calcualtion of forces. When you have confirmed that the configuration if warning-free, just comment out the `relax_geometry` for aimsChain calculation.

You can also set `sc_iter_limit` to a lower value, which should still be much higher than the normal number of cycles that your system consumes. As the path evolves, it can be the case that during a few iterations the geometry become non-physical, and its scf cycle will never converge. Such cases often recover itself within a few iteration, and will not affect the final result. However, a default limit of 1000 will consume lots of computational power in these cases, which is a complete waste especially for larger systems.

The control file should not have `relax_geometry`, any molecular dynamic keywords, and etc. The geometry should not be altered by aims. It is advised that you always use a light setting for the first run, so that tunning settings and confirming convergence can be done efficiently. If a more accurate result is desired, you can submit a tight run using the resulting geometries from the light run, which is always faster than a direct tight run from linear interpolation.

To ensure efficiency, you should configure your control file so that each single point calculation takes at most few minutes to finish. If a tighter setting is desired, consider using `climb_control` for a different control file during climbing image.

aimsChain control file

It is unlikely that you will need a long list of keywords in your control. Tags should be added gradually if you find that the default settings is not sufficient for your purpose. Keys that should be included on the first run are `run_aims`, `initial_file`, `final_file`, `n_image`, `force_thres`, `restart`, and `external_geometry` if so desired. Simply grabbing the control from the samples will also work most of the times.

If you believe that your system follows a rather straightforward path geometrically(i.e. diffusion, small rotation, etc.), you can set the climbing image on your first run and see if it works out correctly. For any complex paths, such as non-trivial isomerization, fine tunning of the control file can be required for the system to converge.

job script

The job script should contain a call to `runchain.py`, as well as loading the necessary modules if that is not done in `.bashrc`. No post processing should be included unless

you are certain that the system will converge and terminate within the time limit.

external geometry

If you have any information about the path you are trying to find, please include them here. This can be a guess for transition state, geometry reported by papers, or your own interpolation by tweaking atoms in a visualization software. Coupled with `resample`, any number of external geometries can help speed up the calculation process, making the process very flexible.

It is a common practice to take the intermediate path from a previous calculation (perhaps before it has gone wild) and use them as the initialization path for a new job. This is perfectly fine, and aimsChain outputs intermediate paths just for this purpose. However, when doing so please remember to remove the first and last image from the path, which are the same geometry as the initial and final states. The list of external geometries should only contain intermediate images, entering initial/final states in there will most likely lead to a dead end.

growing string method

GSM offers another possibility for no-trivial calculations. If you believe your geometry will result in a geometrically complex reaction path, then using GSM will be your best bet.

GSM works by starting from the two end point, and gradually adding images toward the centre of the path. Whenever an image is added, it will be evolved until its forces falls below a threshold. This way we are not requiring any a priori knowledge on the intermediate path, where the standard SM/NEB method approximates with a linear interpolation.

test the interpolation

One of the most important step you can take to ensure a successful MEP evaluation is to ensure that the initial path is physical. This is especially true for periodic systems where the periodic boundary condition plays an unwanted part in this.

We have provided a tool in `aimsChain/tools`, the `interpchain.py`. This gadget performs the resampling and interpolation process exactly the same way that the actual script is doing it, and therefore is a good way to check if the initial path is reasonable.

When ran, the program will create the `interpolation` directory in your project directory (and will clear that folder if it already exists!).

```
-Project directory
|
|--interpolation
     |
```

```
|-image001.in
|
|-image002.in
|
|-...
|
|-path.xyz
```

There will be `n_images` +2 aims geometry created, including initial and final geometry. A multi-frame *.xyz* file is also created in the directory for use with visualization softwares that doesn't support multi-file animation. If the geometry is periodic, the *.xyz* file is repeated in each lattice vector according to `xyz_lattice` to make visualization easier. (because the standard xyz does not encode periodic information) Always check the interpolation if you are working with a new project, which can save you lots and lots of time if you happen to spot a bad interpolation(as shown in sample 4).

a note on periodic systems

Running aimsChain on periodic systems requires extra precaution, because initial files from a periodic system can often provide misleading initial path.

The default interpolation algorithm when `periodic_interpolation` works as follows. For each atom in the final geometry, the coordinates are offset by each lattice vector in both the positive and negative direction. The results are compared with the same atom in the initial geometry, and the coordinate with the shortest distance is used as the actual coordinate for the final atom. This method is good in general.

However, it's still possible to imagine cases where a longer path is the actual path. For example, when trying to simulate the effect of an edge on one end. The shortest path may corresponds to climbing over the edge to reach the other end, while the true path, moving in the other direction, is to simply walk away from the edge.

In these cases, the coordinate of the final geometry must be adjusted manually, so that its coordinate corresponds to its true coordinate after the move, and does not involve any periodic boundary condition. `periodic_interpolation` should be turned off, and the interpolated path should be looked in details to ensure that the atoms in the base are also correctly interpolated.

4.8.5 Running the script

When the script is running, it will first generate a few directories and files in the project directory.

```
-Project directory
|
|--forces.log
|
```

```
 |--climbing_forces.log+
 |
 |--growing_forces.log*
 |
 |--iterations
 |     |
 |     |-iteration0000
 |     |     |
 |     |     |-aims-chain-node-0.00000
 |     |     | |
 |     |     | |-aims-chain-node-0.00000.out
 |     |     | |
 |     |     | |-control.in
 |     |     | |
 |     |     | |-geometry.in
 |     |     |
 |     |     |-aims-chain-node-0.10000
 |     |     |     |
 |     |     |     |-...
 |     |
 |     |-...
 |
 |--paths
       |
       |-iteration0000
       |     |
       |     |-image001.in
       |     |
       |     |-image002.in
       |     |
       |     |-...
       |     |
       |     |-path.xyz
       |     |
       |     |-ener.lst
       |     |
       |     |-path.lst
       |
       |-iteration0001
       |     |
       |     |-...
       |
       |-...
```

+only generated if climbing image is used
*only generated if growing string method is used

Warning! Directories named `optimized`, `paths`, and `iterations` may be cleared/changed if they already exist. Don't leave useful information there!

The `iterations` directory is where all the aims single point calculations are done. Each iteration has its own directory, where each single point calculation is again put in its own directory. They are labelled by the unique id of the particular images as well as the iteration it is in. This is also the place where aimsChain stores optimizer and restart related files.

The `paths` directory contains useful information for you while aimsChain is running. A directory is created for each iteration, containing all the necessary information you might be interested in. The `geometry.in` for each image is written, including the initial and final state. This is the perfect place to extract intermediate path if you would like to start a new job from there. Remember not to include the initial and final geometry in your external geometry list. A *.xyz* animation is written, governed by `xyz_lattice`, which is handy for visualizing the current stage. `ener.lst` stores the current energy of the system, which is extracted from the aims outputs. The energies are offset so that the initial state is at the zero point. This will give a rough idea of the energy landscape along the path at the current iteration. `path.lst` lists the corresponding path in the `iterations` directory, in case you want to check the aims output for diagnostics.

`ener.lst` and `path.lst` also labels the state each image is in. A "FIXED" label indicates that the images is not been moved from iteration to the next, which can be the case for the initial and final geometry, as well as during climbing image calculation. A "CLIMB" label indicates that this image is the climbing image, and when converged, represents the geometry of the saddle point. Normal nodes are simply labelled "Normal".

There are also files named `forces.log`, `growing_forces.log`, and `climbing_forces.log` in your project directory, which records the residual forces in the system for each iteration. They are the most straightforward way to check the current state of the system.

When the path is converged, the result will be put into a directory named `optimized`. They will have the same geometry as the last iteration in the `paths` directory, but they are copied from `iterations` directory so that the aims outputs are included. The same is down for growing string method. Once the growing process is completed all the relevant info are write to the `grownstring` directory, so you don't have to re-grow your geometries for different calculations.

4.8.6 Tips & Guides On Running

It is important to inspect the calculation once in a while when it's running, so that it can be stopped when going astray.

If the forces are going to extremely big values (several hundreds) after reaching a relatively small value, please have a look at the most recent geometry in `paths` to see if it's physical. Any non-physical path is most likely to have been caused by the BFGS optimizer. You should go backward and find the last iteration at which the geometries are physical (most likely corresponding to a small residual forces), and use that as the initial geometry for a new calculation with non-global FIRE optimizer or trm optimizer.

If the forces goes up and down repeatedly, try switching off the global optimizer. If that doesn't help, try FIRE or trm.

When climbing image is not converging, your force_thres might be too large for the system. If you are using climb_mode 1, consider extract the converged path before climbing started, and experiment with mode 2. Otherwise start a new calculation with the converged path and a higher threshold.

If you are focusing on achieving a very precise calculation, and your system is not computationally intensive, you may want to do the following. Increase number of images used. Utilize climb_control for tighter convergence at climbing image. Use climb_mode 2 or 3 for climbing image. Start with non-global FIRE in the first place to avoid potential problems with BFGS.

If you are working with systems that are computationally expensive for aims, try these. Decrease the number of images, but use at least 3. (or more, if you have a complex reaction) First converge a path to a reasonable value.(below 1 $eV/\text{Å}^2$ for example) Extract the path and try aims_restart . Your system may be one of the few that has a consistent wave function configuration as the string evolves to MEP. If the job stops after one iteration, then that has failed, simply re-submit the job after turning the flag off. Set restart in both control.in and chain.in, even if you are not using aims_restart . This will save time when your calculation is stopped when exceeding the time limit. Try to use the lightest setting possible for the calculation. Only switch to tight settings after a good result is achieved on the light setting.

When the climbing image is converged, the image that's labeled "CLIMB" in ener.lst is the transition state. It may be the case sometime that when using climb_mode 1, the converged transition state has a lower energy than its neighbour. This is not a problem in general, its neighbour was not well converged to the path. If you are worried about it, set a tighter convergence threshold or using a different climb mode will help.

4.9 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 [145], well-tempered metadynamics [15], umbrella sampling [228, 143, 204], Jarzynski-equation based steered molecular dynamics [120, 54]. 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 I.2). In the future a patch for modifying FHI-aims in order to compile it with PLUMED will be available on the PLUMED webpage.

4.9.1 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.

4.10 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.

4.10.1 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 form 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 form the user is required here, too.

  - select the total number of slots with the keyword "`# $ -pe impi`", according to the number of replicas. For performance reasons only, it is a good idea to have the number of slots be a multiple of the number of replicas (`n_rex` in `geometry.in.rex`). Informations and warnings concerning this issue will be written to `log_rex`.
  - 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
  - set the number of slots per node (host) with `ncpupn=<#SlotsPerNode>`. This is particularly important for the right distribution of available slots. For performance reasons only, it is a good idea to have the number of slots per replica be a multiple of the number of slots per node (`ncpupn`) or vice versa. Informations and warnings concerning this issue will be written to `log_rex`.

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'
# number of CPU per node (host)
ncpupn=<#SlotsPerNode>
#####################################################################
```

- `run_rex.sh` is a bash script in order to run locally
  - serves as a substitution for `submit.rex` if the SGE is not available
  - if possible, use `submit.rex` because of performance reasons due to the more sophisticated distribution of jobs over the available slots (CPU)

4.10.2 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 attempeted 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 (el.+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 attempeted 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

- It is also possible to restart (prolong) a job that has been completed successfully, i.e. after the desired number of Replica Exchange steps has been performed. In order to do so, set 'type=restart' in submit.rex, set 'MAX_steps <#MaxSteps>' in control.in.rex according to the (new) desired maximum number of steps, and replace the third number in rex_par with that same number '<#MaxSteps>.

- 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.

4.11 Formation energies of charged defects

The Gibbs free energy of formation of a defect is given by

$$\Delta G_f^D = E_{tot}^D - E_{tot}^{perf} - \sum_i n_i \mu_i^{ref} + q\varepsilon_F^{ref} - \sum_i n_i \Delta\mu_i + q\Delta\varepsilon_F, \qquad (4.15)$$

where E_{tot}^D and E_{tot}^{perf} are the total energies of the defected and the perfect sytem, 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 μ_i^{ref}, $\Delta\varepsilon_F$ is the Fermi level referenced to ε_F^{ref} and q is the charge of the system.
A common choice as a reference for the electron chemical potential ε_F^{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:

$$\sum_i^{N_{atoms}} Z'_i = \sum_i^{N_{atoms}} Z_i - q.$$

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 Me_xO_y surface is to distribute the charge uniformly on the metal atoms Me by changing their nuclear charge from Z(Me) to

$$Z(Me^{VCA}) = Z(Me) - \frac{q}{N(Me)},$$

where N(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

$$\Delta\varepsilon_{\mathsf{F}} = (\varepsilon_{\mathsf{F}} - \varepsilon^{\mathsf{D}}_{\mathsf{core}}) - (\varepsilon^{\mathsf{perf}}_{\mathsf{VBM}} - \varepsilon^{\mathsf{perf}}_{\mathsf{core}}).$$

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 $\Delta\mathsf{G}^{\mathsf{D}}_{\mathsf{f}}(L) \approx \mathsf{a}\frac{1}{L} + \mathsf{c}$ for sufficiently large supercells [158]. 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 $\mathsf{L}^{(0)}$. For differently shaped unit cells with lattice constants $\mathsf{L}^{(0)}_1$, $\mathsf{L}^{(0)}_2$, $\mathsf{L}^{(0)}_3$ set for example L:=L_1 and build supercells $L_1 = \mathsf{n}\cdot\mathsf{L}^{(0)}_1$, $L_2 = \mathsf{n}\cdot\mathsf{L}^{(0)}_2$, $L_3 = \mathsf{n}\cdot\mathsf{L}^{(0)}_3$ with integer n. The desired formation energy of a single defect in an infinite supercell $\Delta\mathsf{G}^{\mathsf{D}}_{\mathsf{f}}(L \to \infty)$ 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 [158, 71].

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 and partial atom-projected density of states. 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 references [10, 238, 13]:

> 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).
>
> J. Wilhelm, M. Walz, M. Stendel, A. Bagrets, and F. Evers, "Ab initio simulations of scanning-tunneling-microscope images with embedding techniques and application to C58-dimers on Au(111)." Phys. Chem. Chem. Phys. **15**, 6684 (2013).
>
> A. Bagrets, "Spin-polarized electron transport across metal-organic molecules: a density functional theory approach." J. Chem. Theory Comput. **9**, 2801 (2013).

Please, cite the above works 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 0.1
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
 -eend  <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. [10, 66], if atomic clusters introduced to model parts of metallic electrodes are large enough, the reservoirs can be modeled by

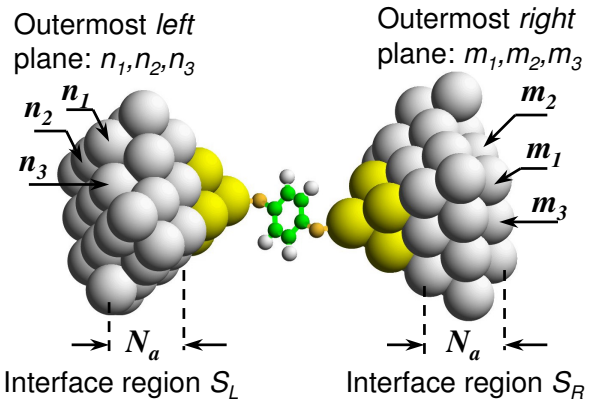


Figure 5.1: A schematic view of the "extended molecule". The shaded regions are interfaces to the two reservoirs. Within the interface regions absorbing boundary conditions are active and the self-energy Σ is introduced. A user defines interface regions by specifying the outermost *left* and *right* 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 $\mathcal{S}_L$ and $\mathcal{S}_R$ (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 μ and μ' label corresponding internal degrees of freedom (i.e., atom-centered basis functions). Here absorption rates η_n are allowed to have non-zero weights only within the interface regions $\mathcal{S}_L$ and $\mathcal{S}_R$. 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
$ecp     on
$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 Σ, 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 forth columns, referring to transmission in up-spin (α) channel and down-spin (β) channel, respectively. A contribution to conductance due to spin σ 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" `fortan`-like format: this number is a local leakage rate η_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 η_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.

5.3.5 Further option: local density of states

Besides a calculation of the ballistic transmission function, you may use AITRANSS to output local density of states projected on groups of atoms forming a molecular junction. This option is controlled by a keyword $ldos and described in detail in the subsequent 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 flag

Purpose: mandatory keyword; flag should take values either on or off. If flag is on, ballistic transmission function is computed. If flag is off, calculation of the transmission function is not performed. In this case you, however, may wish to compute atom projected local density of states that is controlled by the keyword $ldos .

Tag: **$ldos** (tcontrol)

Usage: $ldos flag

Purpose: optional keyword; flag can take values on or off. If flag is on, atom projected local density of states (LDoS) is computed; otherwise (flag is off) calculation of LDOS is not performed.

On output, the (energy dependent) density of states of the system is projected onto atomic orbitals of atoms marked by the same chemical symbols and is redirected to corresponding files, which have the self-explanatory names, e.g. ldos.au.dat, ldos.c.dat, ldos.h.dat and ldos.s.dat in the case of benzene-dithiol molecular junction shown in Fig. 5.1. For example, the file ldos.c.dat would contain LDoS summed up over all six C atoms of the molecule. Furthermore, only LDoS projected onto those electrodes' atoms of the "extended molecule" which are not part of the interfaces to the reservoirs (shown in grey color in Fig. 5.1) is redirected to the file ldos.au.dat.

Data in the output files are arranged in columns as indicated in the file's header, namely:

energy in Hartree (atomic) units; energy in eV units given with respect to the Fermi energy; LDoS (in 1/eV units) in the α spin channel; if open shell calculation is chosen, next column contains LDoS (in 1/eV units) in the β spin channel; last column contains LDoS summed up over two spin channels.

There is a possibility to arrange atoms having the same chemical symbol in groups by using the sixth field of the line staring from `'atom'` as appears, e.g., in the `geometry.in` file (see also a keyword `$coord`). For example, LDoS projected on the two Au atoms indicated by the mask 'apex' as shown in the example below,

```
atom     0.1424445     0.1134435     4.4557346     Au    apex
...
atom    -0.1689195     0.0042317    -4.4601905     Au    apex
```

would appear in the file `ldos.au_apex.dat`. *Caution:* maximum 16 characters can be used to designate a group of atoms.

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 α (up-spin) and β (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).

Tag: `$ecp` (tcontrol)

Usage: `$ecp flag`

Purpose: optional keyword, `flag` can take values `on` or `off`. Default (and recommended) value is set to `on` and serves to substantially accelerate electron transport calculations.

In that case, exploiting Green's function formalism, "core" electronic states of atoms with $Z \geq 19$ are integrated out and projected on a subspace of the remaining ("valence") electronic states. Thus, dimension of the effective Hilbert space of the "extended molecule" is reduced (see `$valence_electrons`), and only valence states are coupled to macroscopic reservoirs via model self-energies. For atoms from the n-th period of the periodic table, core states are associated with those "atomic" basis functions, which have the principle quantum numbers up to $n - 2$.

Tag: `$valence_electrons` (tcontrol)

Usage: `$valence_electrons` N_{val}

Purpose: optional keyword, N_{val} is integer number, which is evaluated automatically and specifies amount of valence states in the "extended molecule" when a keyword `$ecp` is switched on.

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 η_1
$s2i η_2
$s3i η_3

Purpose: positive real numbers η_1, η_2 and η_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 diffent 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-none 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 every 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 opertaing system vendors limit the default user stack size to ≈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 I'm running a calculation for a large system, and it exits abrutply. What's going on?

It is possible that you are indeed running out of memory. See the next section, "What do I do if I run out of memory?"

However, do check if you really did set "ulimit -s unlimited" before running the calculation. If FHI-aims is compiled with an additional C compiler (this is as simple as defining the "CC" environment variable in `make.sys` or in the `Makefile`), then FHI-aims' standard output also writes the stack size that is set for each MPI task. Sometimes the result can be surprising – for example, some MPI libraries' `mpirun` command does not propagate the stack size limit to all compute nodes.

A.6 What do I do if I run out of memory?

Like all other electronic structure codes, the current bottleneck for FHI-aims when performing calculations for large systems is memory usage. A number of flags exist to alleviate memory usage, but as many of them have an associated trade-off of performance overhead, poor scaling for small system sizes, or breaking post-processing techniques, they have not been enabled by default. For large systems where memory usage becomes an issue, however, enabling them will be worth it. This list of flags may be found at in Section 3.38, "Large-scale, massively parallel: Memory use, sparsity, communication, etc."

For hybrid functionals please make sure that you really need the basis set that you are requesting. For example, the tier 2 basis sets specified by tight and really_tight settings for light elements can be far too large. You may be able to get away with "intermediate" settings instead. Especially for hybrid functionals, large numbers of basis functions per atom really increase the time and memory consumption much more drastically than for semilocal DFT.

Please ask regarding intermediate settings in our forum – especially for hybrid functionals or for many-body perturbation theory. It's worth it.

A.7 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 expressable as linear combinations of some others. The eigenvalue problem Eq. (3.27) 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 eigenvakue 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 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. (3.27).

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.8 No convergence of the s.c.f. cycle even after many iterations

One first thing first: If you encountered unexpected s.c.f. convergence issues, did look at your exact `geometry.in` file in a viewing program, such as jmol? One of the most common reasons to find unexpectedly bad s.c.f. convergence for apparently harmless structures are issues such as the wrong lattice parameter for the structure in question, an atom in the wrong location, or the wrong structure altogether. Structures that are chemically unstable are often not happy at all when it comes to the stability of their electronic structure, and slow or no s.c.f. convergence can be an indicator of a simple geometry mistake.

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.

Visualizing the actual s.c.f. convergence behavior of your run can be a first step to success. See Sec. 3.10.1 for a brief description of how to do this.

The `adjust_scf` keyword now automatically controls s.c.f. convergence. It may be best to first remove *all* scf-convergence related keywords from `control.in` and see if this works. Sometimes, problems arise from mis-setting such keywords. As a result, for example we no longer recommend setting *any* of the `sc_accuracy_*` keywords explicitly.

If a `control.in` file without any scf-convergence related keywords does not solve the problem, the next steps are to adjust the available keywords carefully and step by step.

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.

A first line of defense against bad s.c.f. convergence is the `sc_init_iter` keyword. This keyword resets the Pulay mixer completely after a specified number of s.c.f. iterations. This is done only for the first s.c.f. cycle and can dramatically improve the covergence behaviour of problematic cases. The current default (as of this writing) is for the mixer to reset itself after 1001 iterations.

Typical settings that work for wide varieties of systems are as follows:

- Semiconducting or insulating solids or molecules (i.e., insulating systems with an appreciable band gap):

```
mixer pulay
charge_mix_param 0.2
occupation_type gaussian 0.01
```

are usually enough to work for such systems.

- Metallic systems with no appreciable band gap, slabs to model surface phenomena, etc.:

```
mixer pulay
charge_mix_param 0.02
occupation_type gaussian 0.1
```

or similar should usually work. Note that `charge_mix_param 0.02` is *not* a small value for a mixing parameter, since this is employed with a Pulay mixer. After several iterations, the Pulay mixer has usually figured out what the right mixing parameters should be and thereby tends to undo the slowing effect of a small mixing parameter.

- Periodic slab models that are still problematic:

In periodic systems, FHI-aims uses a Kerker preconditioner by default, in addition to the usual Pulay mixer. For very anisotropic metallic systems (think graphene sheet with a 10x10 unit cell and a large vacuum), the Kerker preconditioner may not be appropriate – switching it off may help:

```
preconditioner kerker off
```

- Slow convergence for spin polarized systems:

Only ever run a spin polarized calculation if there is a good reason to do so. Running a clearly non-spin-polarized system with `spin none` just for aesthetic reasons is a bad idea: It will double your computer time usage in the best of cases, *and* it will require you to begin the calculation from some finite spin initialization that may not be close to the self-consistent electronic structure you are seeking. Thus, spin polarization may add additional s.c.f. cycles *and* cost extra time in any case.

Obviously, please do run spin polarized calculations if there is a good physical reason to do so. Just do not underestimate the cost. It is also wise (= essential!) to think about the initial moments chosen for the initialization of the calculation. The closer the initial moment distribution matches the expected self-consistent result, the better (faster) the convergence.

In contrast, beginning every spin-polarized calculation with a high spin state for every atom may be a highly bad idea. In particular, do not run molecular or condensed phase calculations (i.e., anything other than single free atoms) with `default_initial_moment hund`. Hund's rules, which can be read up on Wikipedia and a host of other sources, apply to free atoms. Condensed systems have very different spin moment distributions.

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 metalic 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 hard cases, we also recommend to review in detail all the options available in Sec. 3.10; better yet, contact us (see Section 1.7).

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-tothed 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

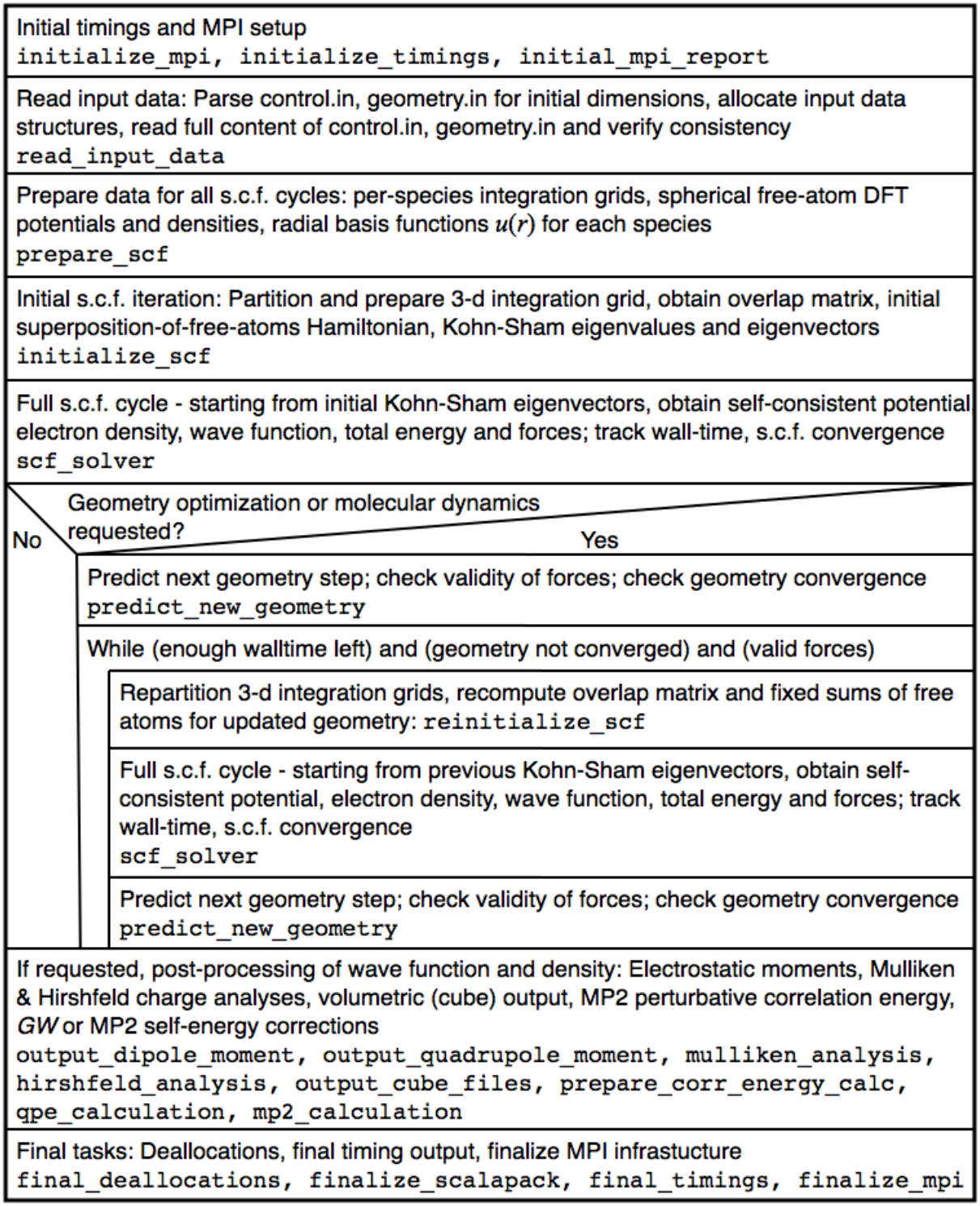


Figure B.1: High-level program flow of FHI-aims

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. (3.57) (see Sec. 3.23 for details) and the Coulomb matrix of the auxiliary "resolution of the identity" basis set [denoted $V_{\mu\nu}$ in Eq. (3.57)] are precomputed. The most important task in `initialize_scf.f90` is the initial solution of the Kohn-Sham equations Eq. (3.27), 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 $\delta\tilde{n}_{\mathrm{at},lm}(r)$ [see Eq. (Eq;mp)], and construction of the multipole components of the Hartree potential, $\delta\tilde{v}_{\mathrm{at},lm}(r)$
4. construction of the full electrostatic potential $v_{\mathrm{es}}(\boldsymbol{r})$ on all points $\boldsymbol{r}$ 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. (3.27), 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.

A current set of "code conventions" is maintained at the Wiki at https://aims-git.rz-berlin.mpg.de. A subset of these recommendations includes:

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.

Again, please consult the https://aims-git.rz-berlin.mpg.de wiki for the full and current recommended code conventions.

Appendix C

Debug Manager - a centralized debugging facility for developers

FHI-aims features a centralized utility for controlling debug output. This utility keeps track of all registered modules (in a physical sense, e.g. the DFPT features are a module in this context) and can enable their debug output selectively via the *control.in* file.

The main idea behind this utility is to provide a simple tool to generate debug-flags that can be set in the *control.in* without having to register them manually in the input parsing routines. Furthermore, it also provides a central debug interface to the user, instead of having to use different flags, flags that can only be toggled in the sourcecode or even no toggles at all.

You can enable debugging for any registered module in the input file by adding a `debug_module my_module`. The list of currently supported modules can be found in the file *init_debug.f90*.

To register your module, you only need to add a single line to the *init_debug.f90*:

```
register_debugmodule("your_tag")
```

Afterwards, you can conveniently activate debugging in the *control.in* by using the flag:

```
debug_module your_tag
```

This tells the code to enable debugging for your code. Just call

```
debugprint(message, your_tag)
```

in your code to only print the message if debugging for your module is enabled. For more complicated debug functionality, there also is

```
module_is_debugged(your_tag)
```

This function returns a logical value and thus can be used as condition in an enclosing if-block. Both functions are located inside the `debugmanager` module, which you can import like any other module with a use-statement.

Appendix D

XML output

The FHI-aims can print results of the calculations in an XML format. This functionality is provided by the `xml_write` module. The module is quite general and capable of printing any XML, but also has convenient functions for printing scalars, vectors, matrices and n-dimensional arrays. The formatting of the data in the XML elements is both human-readable and machine-readable. The obvious advantage is that such an output from the code is extremely easy and fast to parse.

There are two ways how the XML module can be used. There is a global XML file which can be turned on with `xml_file <filename>` in `control.in`. Printing to this file is as simple as loading the module and calling one of its routines. Note however that if the block of code that prints to this global file calls another code also printing to the global file, the XML output of these two codes might get intertwined, resulting in a valid but nonsensical XML file.

A second way how to use the XML module is to create a new "local" XML file with the `open_xml_file` routine and use that one by calling the module printing routines with the optional argument `file=<xml_file_instance>`.

Appendix E

Optional Libraries to be Linked into FHI-aims

A core principle for FHI-aims is that it should be possible to build all essential functionality with only a minimal set of dependencies, typically a Fortran compiler, BLAS and LAPACK libraries, an MPI library, and the ScaLAPACK library. These packages are essential for performance and usually available on every relevant supercomputer architecture.

We note that while it is often easy to fulfil other, far more complex dependencies on standard Linux type systems with household tools, this is not always true for highly specialized supercomputers. Yet, that computer may be precisely one the multi-million dollar machine which would help solve an advanced scientific problem. It is essential that a mere human be able to compile FHI-aims with the appropriate performance on such a machine.

That said, it may be beneficial to connect other libraries to FHI-aims for special purposes, including libraries which require cross-compilation with a C compiler. This section is intended to be developed into a list of such libraries. Please also see Section I.1.1 for some basic information.

E.1 Adding Optional Libraries into FHI-aims: Stubs

The optional presence or absence of certain libraries from FHI-aims at compile time means that the main FHI-aims code must be able to deal with the possible absence of certain subroutine or function calls to those libraries at compile time.

In many codes, this is handled by adding preprocessor statement. In FHI-aims, preprocessor flags are **NOT** the way forward. The simple reason is that a single preprocessor flag may remain understandable for an ordinary user; the proliferation of ten or more preprocessor flags will create a serious obstacle for others to obtain a reasonably compiled code. Code that introduces preprocessor statements into FHI-aims will be removed from the code base.

In FHI-aims, the recommended way to optionally add or circumvent a given library is by

creating a "stub" file for those library calls – essentially, empty subroutines that inform a user that they should not have ended up here with a given version of the code (i.e., without linking to a certain external library). At compile time, an appropriate flag in the Makefile and Makefile.backend should be created that switches between the real external library (if available) or the "stub" file. Please follow the example of "libxc" etc. in the "CMakeLists.txt" file, the Makefile, and Makefile.backend for more information. The principle is simple and easy to apply for any other optional library.

E.2 Spglib

Spglib is a library for finding and handling crystal symmetries, written by Atsushi Togo (**http://spglib.sourceforge.net**). FHI-aims can be interfaced to the spglib.

So far, FHI-aims supports the determination of the crystal symmetry for a given (periodic) geometry file and k-point reduction based on this analysis (local and semi-local functionals only). Theoretical background and keywords are described in Section 3.40.

Prerequisites:

spglib is written in C and therefore needs a C compiler with appropriate options in addition to the usual Fortran compiler with which FHI-aims is built.

Ideally the spglib source files should be placed in the folder `external/spglib` contained within the `FHI-aims` source (src) directory.

Tag: `use_symmetry_analysis` (control.in)

Usage: `use_symmetry_analysis .true. / .false.`

Purpose: This flag activates the interface to the spglib to obtain symmetry information for the provided geometry file once. The symmetry information is directly written after the geometry printout in the FHI-aims standard output. spglib needs to be compiled with FHI-aims to get the output.

Default: `.true.`

Tag: `sym_precision` (control.in)

Usage: `sym_precision value`

Purpose: This value determines the accuracy for identifying symmetric atoms. The lower the value, the more strictly the atomic positions must overlap.

Default: 10^{-5}

E.3 Libxc

Libxc is, accordingly to its website, "a library of exchange-correlation functionals for density-functional theory. The aim is to provide a portable, well tested and reliable set of exchange and correlation functionals that can be used by all the ETSF codes and also other codes." More information may be found at http://octopus-code.org/wiki/Libxc. The version distributed with the FHI-aims source code is 4.0.2.

Compiling FHI-aims with Libxc provides opportunity to perform SCF calculations with either the internal FHI-aims routines or the large repository of options available in Libxc. Libxc support is beneficial for extended functionality in FHI-aims, such as calculations of NMR parameters and the atom_sphere radial solver. If a particular method requested by the user requires Libxc, the calculation will stop and inform the user to re-compile with Libxc support. Most users running standard DFT/hybrid SCF calculations will not need to compile with Libxc support.

Prerequisites:

Libxc is written in C and therefore needs a C compiler with appropriate options in addition to the usual Fortran compiler with which FHI-aims is built.

Specifically for Libxc, include in `Makefile` or `make.sys`:

```
USE_LIBXC = yes
```

E.4 `cffi` — Python 2/3 interface to FHI-aims

`cffi` provides options to execute arbitrary Python code or launch an interactive Python console from within FHI-aims. Various Fortran runtime variables are accessible from the Python environment. At the time of writing, the interface is limited to the grid point batches, electron density, grid partition function Hirshfeld volumes and atomic coordinations. Adding further objects is a fairly straightforward process (see below) and the author of these lines (jhermann@fhi-berlin.mpg.de) will gladly do so upon request.

Tag: `python_hook` (control.in)

Usage: python_hook <label> (<filename> | REPL) <attribute>

Purpose: Register a Python hook to a specified location.

<label> is a string, specifying the location in the FHI-aims run, at which the hook is activated (see below for a complete list).
<filename> is a string, specifying the Python source file that should be executed (details below). If REPL is given instead, FHI-aims launches an interactive Python console at the given location. The latter option is available only in serial runs.
<attribute> is a string, specifying optional attributes of the hook. Currently, this can be only parallel, which specifies that the hook should be run in all MPI tasks. The default is to run only in the root task.
The tag can be specified repeatedly. There can be only one hook registered to a given location, since it can easily invoke other scripts.

Prerequisites

The extension can be linked to aims using both Make. Note that when you compile with Python 2 or 3, you then need to use that particular version in your user scripts.

- To compile with Make, define USE_CFFI=yes. This also requires specifying a Python interpreter with PYTHON=<path to python>. The Python include files and libraries are detected automatically and printed at the top of the Make output.

The interface is written using the iso_c_binding module, which is a Fortran 2003 feature. It is supported by all major compilers.

The build extension links FHI-aims to a dynamic Python library. Under ideal circumstances, everything should work out without any intervention, but manual intervention may be needed in non-standard environments. The Python installation needs to have packages cffi ($\geq$1.5.2), Numpy/Scipy. The mpi4py package is not required, but its absence severely limits the functionality.

Troubleshooting

In Linux, dynamic libraries are usually recorded only by their names during linking and found dynamically at runtime at several standard locations. If you link against your own Python installation, such as Anaconda, you need to make sure that FHI-aims can find the Python library at runtime. This can be done by adding anaconda/lib to $LD_LIBRARY_PATH. Since Anaconda now packages includes its own version of the Intel Math Kernel Library, which could clash with any system MKL, it is recommended to uninstall MKL from Anaconda if you include it in $LD_LIBRARY_PATH.[1]

On OS X, dynamic libraries are supposed to contain their absolute installation path, in which case these paths are recorded during linking and used at runtime. This

[1] A more user-friendly behaviour could be setup using RPATH in the future.

is the case with the system Python and Homebrew Python, but not with Anaconda Python. To use this extension with Anaconda Python on OS X, either set the variable $DYLD_LIBRARY_PATH to the Anaconda library directory anaconda/lib or (better) change the install path of the Python library in the compiled FHI-aims binary with the system install_name_tool. Furthermore, some version of Anaconda Python on OS X do not seem to properly initialize Python paths when embedded, in which case $PYTHONHOME needs to be set when running FHI-aims.

Usage

The Python interface can be used either in an interactive or non-interactive mode. The interactive mode is activated by the option REPL (see python_hook). In this mode, AIMS launches an interactive Python console at a given location. This is available only in a serial run. In the console, connection to the running FHI-aims instance is provided via a local variable ctx. Various quantities are accessible as attributes of this context object. For example,

```
  Self-consistency cycle converged.
  [...]
------------------------------------------------------------
  Executing Python hook: post_scf

There is a local variable ‘ctx‘.
See ‘help(ctx)‘ for details.
Press CTRL+D to
continue the aims run.
Type ‘exit(1)‘ to abort aims.
+>>> ctx
<AimsContext 'rho, batches, coords, partition_tab'>
+>>> ctx.coords
array([[-3.77945226,  3.77945226],
       [ 0.,  0.],
       [ 0.,  0.]])
+>>> ctx.rho
array([[  5.81201802e-02,   3.72428091e-01,   3.72426419e-01, ...,
          1.54065044e-05,   4.86078607e-12,   1.17640761e-29]])
+>>> ctx.rho[:] = 0
+>>>
```

The context object provides also several convenience functions which are documented in help(ctx). When the console is quit normally (CTRL+D), the FHI-aims run continues. When the return code is non-zero (for example with exit(1)), the FHI-aims run is aborted.

In the non-interactive mode, which is available also in parallel runs, the context object is passed to the user-defined function run which is loaded from the specified file. Two

locations are currently available at which the `run` function is executed, as specified by the argument to the `python_hook` keyword:

post_scf immediately after the end of the self-consistent loop, before any post-processing

post_hirshfeld immediately after the end of the Hirshfeld analysis, that is, before any van der Waals routines

A minimal user script (can be both Python 2 and 3) could look for example like this:

```
import json

def run(ctx):
    with open('coords.json', 'w') as f:
        json.dump(ctx.coords.tolist(), f)
```

Note that the provided variables are local to a given MPI process. For synchronization, one can use the `mpi4py` Python package. Some convenience synchronisers are defined in `ctx`. For exampeles of how to use the `mpi4py` package, see the commented implementation of `ctx.gather_all_grids()` in `cffi/python_interface.py`. The user script can optionally define also function `parse`, which is then called without any arguments during parsing of `control.in`. This can be used to verify or precompute certain data before launching the full calculation. Any uncaught exceptions in the both `parse()` and `run()` cause aims to abort immediately. To recapitulate, a user script has one or two entry points: (i) the mandatory `run` is executed at the location specified in `control.in` and takes the single context argument and (ii) the optional `parse` function, which does not take any arguments, and is always called at the end of parsing `control.in`.

At the time of writing, two example hooks are provided in `aimsfiles/utilities/python_hooks/`. The first one plots the electron density in the xy plane. Below is a shortened glimpse to get a general idea, see the original file for the complete implementation.

```
# ~~ imports ~~

bohr = 0.52917721067

def run(ctx):
    points, rho = ctx.gather_all_grids(['rho'])
    if ctx.rank == 0:  # the grids are gathered only on the root process
        points = points.T  # (3, npts) -> (npts, 3)
        rho = rho.sum(0)  # sum over spin
        in_plane = abs(points[:, 2]) < 1e-10  # points in xy plane
        points = bohr*points[in_plane, 0:2]  # filter & take x, y & scale
        rho = np.log10(1+rho[in_plane])  # filter & log scale
        X, Y = np.mgrid[-4:4:400j, -2:2:200j]  # get rectangular grid
        # interpolate density to rectangular grid
        rho = griddata(points, rho, (X, Y), method='cubic')
        # ~~ matplotlib plotting ~~
```

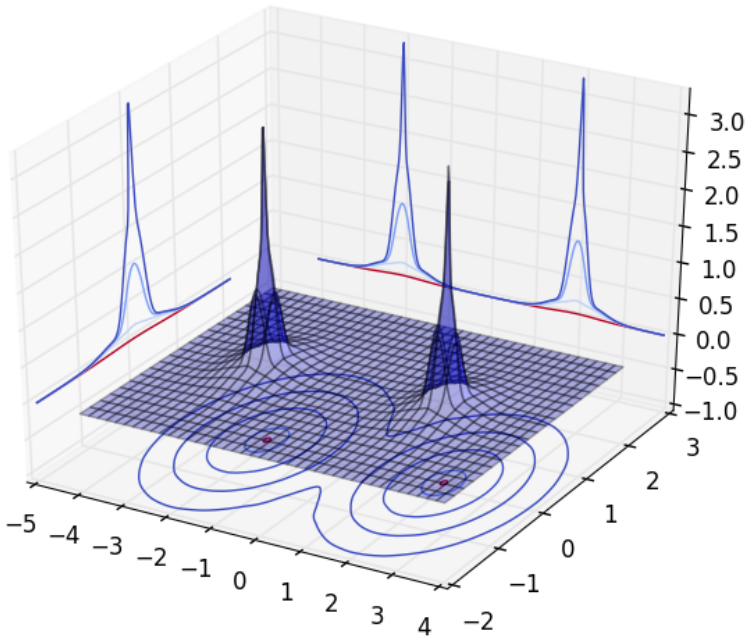


Figure E.1: Electron density of an argon dimer in the xy plane produced by FHI-aims via interface to Python.

Saving this file into `plot-density.py` and putting `python_hook plot-density.py parallel` into `control.in` produces Figure E.1 for an argon dimer.

The other example hook loads Hirshfeld volumes from the output of a previous FHI-aims calculation. The parsing part of the hook is executed when FHI-aims parses `control.in` and searches for the Hirshfeld analysis section in `hirshfeld.out`. The actual modification in all processes happens after the normal Hirshfeld analysis is performed by specifying `python_hook load-hirshfeld.py parallel`.

```
import numpy as np

volumes = None

def parse():
    global volumes  # otherwise volumes would be local to function
    with open('hirshfeld.out') as f:
        while 'Performing Hirshfeld analysis' not in next(f):
            pass  # find the Hirshfeld section
        volumes = np.array(list(get_volumes(f)))

def get_volumes(f):
    for line in f:
        if not line.strip():
            return
        if 'Free atom volume' in line:
            free = float(line.split()[-1])
        elif 'Hirshfeld volume' in line:
```

```
            hirsh = float(line.split()[-1])
            yield hirsh/free

def run(ctx):
    ctx.hirshfeld_volume[:] = volumes  # replace values in-place
```

The two-step operation saves computation time if the parsing ended with an error, since it would abort FHI-aims right after start. By specifying also

```
sc_iter_limit            0
postprocess_anyway       .true.
```

in `control.in`, one can skip the SCF cycle completely and evaluate only the van der Waals part of the energy.

Extending the interface

To extend the range of objects provided by the context object, several simple steps need to be followed.

`python_interface.f90` The Fortran type `AimsContext_t` needs to be extended and the added component needs to be initialized in `get_aims_context()` and potentially deallocated in `destroy_aims_context()` if it required any allocation. The latter is of concern only for mapping Fortran types, not simple arrays.

`cffi/python_interface.h` The corresponding C struct `AimsContext_t` needs to be updated accordingly. When these two steps are done, the object is already available in its raw form in `ctx._c_ctx`.

`cffi/python_interface.py` The initialization of the Python object `AimsContext` in `call_python_cffi_inner()` needs to be updated accordingly. For simple arrays, this amounts to simply wrapping the raw C pointers with Numpy arrays. For wrapping Fortran types, see how `BatchOfPoints` and `GridPoints` are wrapped.

To add new hook labels, add the following lines to the desired location:

```
use python_interface, only: run_python_hook, python_hooks

% ...

if (python_hooks%<hookname>%registered) then
    call run_python_hook(python_hooks%<hookname>)
end if
```

and extend `HookRegister_t` and `register_python_hook()` in `python_interface.f90` by adding `<hookname>`.

Appendix F

Split large cluster allocation to multiple aims instances

Specific tasks need to perform many small jobs rather than a few big ones. The job submission of such tasks on large compute facilities, which typically require at least a few hundreds MPI processes, can be cumbersome. FHI-aims allows for running multiple instances of itself and split a global MPI communicator into smaller, independent parts. This allows for a unified approach to submit one large job and let the splitting happening in aims.

To use this feature, you need to compile the multiaims executable by compiling the multiaims target. When using CMake, this is controled via the `MULTIAIMS` option. When using Makefile, the target `multi.scalapack.mpi` should be used. In either case, a C compiler is mandatory in addition to the usual requirements.

The multiaims environment needs its jobs distributed in subdirectories labeled 1 to N, where N is the total number of jobs. In the main directory one additional control file *multiaims.in* needs to be specified. This file contains two keywords:

Tag: `tasks_per_subjob` (multiaims.in)

Usage: `tasks_per_subjob value`

Purpose: Specifies the number of processes working on one subtasks. Should be a divisor of total number of processes.

Tag: `start_id` (multiaims.in)

Usage: `start_id value`

Purpose: Specifies the first job folder to start with.

Appendix G

GPU Acceleration of FHI-aims

G.1 Introduction

G.1.1 Overview of GPU Acceleration Philosophy in FHI-aims

FHI-aims uses a batch integration scheme[96] in which the real-space integration points are broken up into spacially localized batches of points. Each batch of points is assigned to an MPI rank, and each MPI rank processes its assigned batches sequentially. After all batches have been processed, the MPI ranks communicate the final results to one another.

The batch integration scheme is at the heart of FHI-aims' O(N) scaling in number of atoms for most of the steps of the SCF cycle. (An important exception is the solution of the Kohn-Sham equations, which is handled by the ELSI infrastructure.) Only basis elements that touch an integration point will contribute to the quantity being calculated for a given batch. As basis elements have finite spacial extent, for a sufficiently large non-periodic system or unit cell of a periodic system, the number of basis elements needed for a given fixed-sized batch will be saturated. Adding more atoms to the system, i.e. increasing the size of a non-periodic system or using a larger unit cell for a periodic system, will increase the number of batches linearly, but not the work done per batch, leading to linear scaling in number of atoms.

The batch integration scheme in FHI-aims lends itself naturally to GPU acceleration. The details vary based on the task being accelerated, but the general strategy is:

1. The MPI rank sets up a batch.
2. The MPI rank communicates batch details to its assigned GPU.
3. The GPU performs work on the batch.
4. If the MPI rank needs to process the batch further, the GPU communicates the results back to its assigned MPI rank.
5. After all batches have been processed, the GPU communicates its final results back to its assigned MPI rank.

As each MPI rank processes its batch independent of other MPI ranks, no significant effort is needed to use GPU acceleration in an MPI environment. The batches are small enough that they fit into memory on an NVIDIA GPU. As each batch is statistically similar in size, the memory usage of a given batch is independent of system size; the GPU will not run out of memory as the system size increases for a fixed number of MPI ranks. Furthermore, most of the computation time for tasks utilizing the batch integration scheme is taken up by a small number of BLAS/LAPACK subroutine calls occurring at the end of the batch processing. These subroutine calls can be easily replaced by cuBLAS (https://developer.nvidia.com/cublas) calls.

The pseudocode for this process is:

```
do i_batch = 1, n_batches
  set_up_batch_on_cpu
  copy_batch_information_to_gpu
  call cuBLAS_Function()
  if gpu_data_needed_on_cpu
    copy_partial_gpu_data_back_to_cpu
    cpu_performs_work_on_partial_gpu_data
  end if
end do

copy_gpu_final_data_back_to_cpu
```

G.1.2 Current State of GPU Acceleration in FHI-aims

The steps needs to use GPU acceleration in FHI-aims are:

1. Make sure prerequisites are installed.
2. Compile FHI-aims with GPU support. This may be accomplished by using CMake or Makefile.
3. Add GPU acceleration keywords to `control.in`.
4. Run FHI-aims as normal.

It cannot be stressed enough that the user should consult the documentation for their architecture, as their architecture may require additional steps to use GPU acceleration beyond what is listed here.

The GPU acceleration code is considered stable and suitable for production calculations. An example scaling plot for timings of the first SCF step for 128 atoms of GaAs on Titan Cray XK7 is shown in Figure G.1. We generally find that the charge density update shows the largest GPU acceleration speed-up. Larger speed-ups are observed as the basis set size is increased. If a non-periodic system or unit cell of a periodic system is too small (say, a primitive cell of GaAs running on 32 MPI ranks), a slow-down may actually be observed.

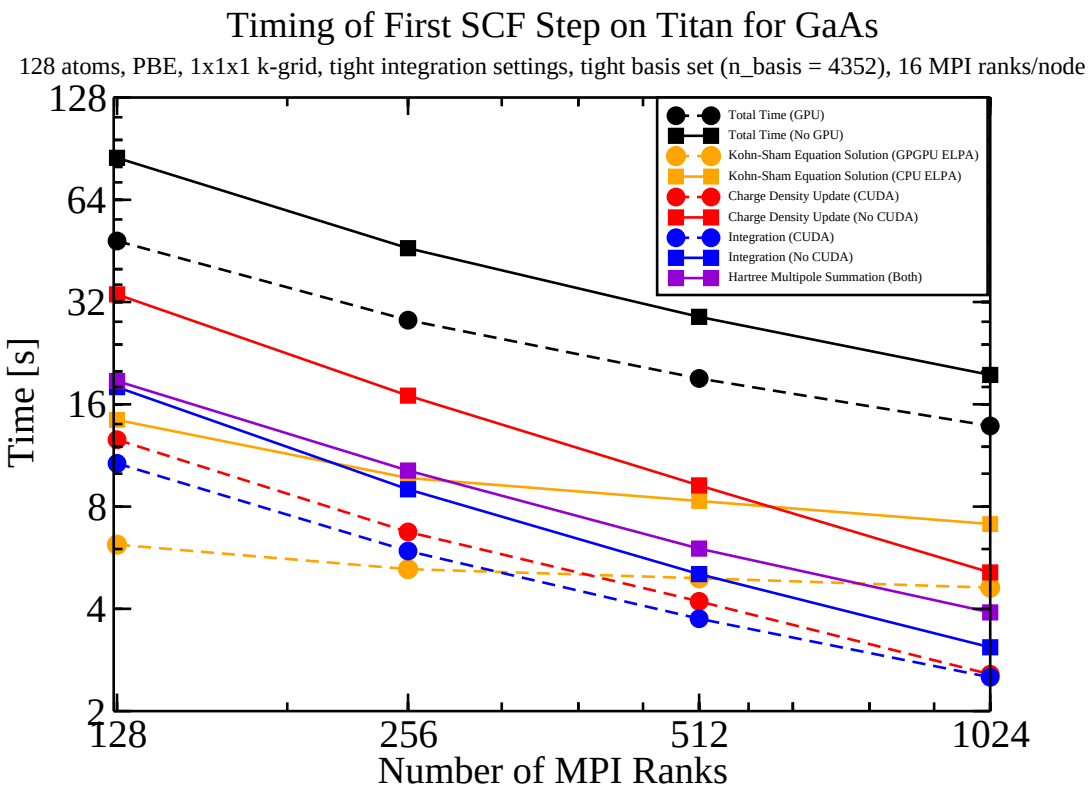


Figure G.1: Example scaling plot for GPU acceleration. The solid lines are CPU-only calculations, and the dotted lines are GPU-accelerated calculations. At present, there is no GPU acceleration in the Hartree multipole summation, so both CPU-only and GPU-accelerated calculations have the same timings for this task.

The list of tasks we have GPU accelerated natively in FHI-aims is:

- Integration of the Hamiltonian matrix
- Charge density update via density matrices
- Pulay forces
- Stress tensor

In the future, we plan to natively GPU accelerate the following tasks:

- Hartree multipole summation
- Construction of the Fock matrix (for Hartree-Fock, hybrid-functional, and beyond)

G.2 Prerequisites

The CUDA Toolkit package is needed to enable GPU acceleration in FHI-aims. Supplied by NVIDIA, CUDA may be downloaded free of charge at https://developer.nvidia.com/cuda-toolkit. It is recommended to use the NVIDIA Multi-Process Service (MPS, see https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf).

The GPU acceleration in FHI-aims requires at least CUDA versions 10.1.

G.3 Installation

Compiling the FHI-aims executable with support for GPU acceleration requires CMake ≥ 3.8. A set of GPU acceleration flags should be included in the CMake initial cache file. The installation procedure is otherwise the same as a standard FHI-aims installation. We have only tested the GPU code for the scalapack.mpi target.

G.3.1 Example `initial_cache.cmake` file for GPU Acceleration

Note The example below, created for 'node18' of the Timewarp cluster at Duke university, covers the GPU-related flags in `initial_cache.cmake`. It may be used as a template for the user's `initial_cache.cmake`. We encourage users to put their own GPU acceleration compilation flags on the "Known compiler settings" section of the FHI-aims GitLab wiki (https://aims-git.rz-berlin.mpg.de/aims/FHIaims/wikis).

Architecture of Timewarp 'node18':

- CentOS Linux 7
- Intel Xeon Silver 4114 CPU 2.20GHz
- NVIDIA Titan V100
- Intel Fortran 18.0.2
- CUDA 10.1
- CMake 3.17.3

`initial_cache.cmake`:

```
set(CMAKE_C_COMPILER icc CACHE STRING "")
set(CMAKE_C_FLAGS "-O3 -ip -fp-model precise -DNDEBUG -std=gnu99" CACHE STRING "")
set(CMAKE_Fortran_COMPILER mpiifort CACHE STRING "")
set(CMAKE_Fortran_FLAGS "-O3 -ip -fp-model precise" CACHE STRING "")
set(Fortran_MIN_FLAGS "-O0 -fp-model precise" CACHE STRING "")
```

```
set(USE_CUDA ON CACHE BOOL "")
set(CMAKE_CUDA_FLAGS "-O3 -DAdd_ -arch=sm_70" CACHE STRING "")

set(LIB_PATHS "$ENV{MKLROOT}/lib/intel64 $ENV{CUDA_HOME}/lib64" CACHE STRING "")
set(LIBS "cublas cudart mkl_scalapack_lp64 mkl_blacs_intelmpi_lp64
    mkl_intel_lp64 mkl_sequential mkl_core" CACHE STRING "")
```

- How to determine the `-arch` or `-gencode` CUDA flags? Your CUDA installation should come with a utility called deviceQuery, which is located in `samples/1_Utilities/deviceQuery` in the CUDA root directory. Copy that directory into your work directory, enter, and build. If you get any build errors, edit the Makefile accordingly. When successful, run the executable `deviceQuery`. The line "CUDA Capability Major/Minor version number" contains the relevant information. For example, if it says 6.0 then use `sm_60` with the `-arch` or `-gencode` flags.
- How to choose between multiple GPU cards installed on the system? Use the environment variable `CUDA_VISIBLE_DEVICES`.
- When using gfortran with CMake version ≥ 3.8 and < 3.11, it can happen that the executable wants to link to the wrong libgfortran library at runtime. This can be prevented by pointing `CUDA_LINK_DIRS` to directories that contain the CUDA libraries (e.g., "/opt/cuda/lib64/stubs /opt/cuda/lib64"). For more information, see this: gitlab.kitware.com/cmake/cmake/issues/17792.

G.3.2 Example `initial_cache.cmake` file when using HIP (EXPERIMENTAL!)

This subsection gives an example case of `initial_cache.cmake` when using HIP. This example was tested on node 'nid005000' of LUMI clusters GPU Early Access Platform (EAP). Compiling the FHI-aims executable with HIP support requires CMake ≥ 3.21. This example uses ROCm v5.1.4.

Architecture of LUMI EAP 'nid005000':

- SLES 15.3
- AMD EPYC 7A53 CPU 2.1GHz
- AMD MI250
- GNU Fortran 11.2.0
- HIP 5.1.4
- CMake 3.23.2

`initial_cache.cmake`:

```
set(CMAKE_Fortran_COMPILER "ftn" CACHE STRING "")
set(CMAKE_Fortran_FLAGS "-O2 -ftree-vectorize -funroll-loops
     -fallow-argument-mismatch -ffree-line-length-none" CACHE STRING "")
set(Fortran_MIN_FLAGS "-O0 -ffree-line-length-none" CACHE STRING "")
set(CMAKE_C_COMPILER "cc" CACHE STRING "")
set(CMAKE_C_FLAGS "-O2" CACHE STRING "")

set(USE_HIP ON CACHE BOOL "")
set(SET_HIP_ARCH "gfx90a" CACHE STRING "" FORCE)
set(CMAKE_HIP_FLAGS "-O2 -DAdd_ -std=gnu++17 " CACHE STRING "")
set(HIP_LINK_DIRS "$ENV{ROCM_PATH}/llvm/lib" CACHE STRING "")

set(LIBS "hipblas" CACHE STRING "")
```

- By default CMake tries to find Clang compiler. If user wants to use some other compiler for example hipcc, then this person can add the following line in to `initial_cache.cmake`:

  ```
  set(SET_HIP_COMPILER "hipcc" CACHE STRING "" FORCE)
  ```

- Do not use `CMAKE_HIP_COMPILER` to set the used HIP compiler, it is not supported in the FHI-aims CMake script. Also the HIP architecture must be set by setting `SET_HIP_ARCH`, otherwise there can be an error.

- The line which sets `HIP_LINK_DIRS` prevents the compiler from linkin against an old version of libgfortran.

- Sometimes `amdhip64` should be added to `LIBS`.

- There is not yet HIP support for ELPA eigensolver.

WARNING This is an experimental release and there are still known issues to be solved. This code is not advised to be used for production runs. Check your numbers!

G.4 Running FHI-aims with GPU Acceleration

Compiling the FHI-aims executable with GPU acceleration support will not automatically turn on GPU acceleration. To use GPU acceleration when running FHI-aims, the user specifies which tasks should be GPU accelerated independently using the `control.in` keywords below:

Tag: `use_gpu` (control.in)

Usage: use_gpu flag

Purpose: Use GPU acceleration methods that are considered stable. flag is optional. It can be either .true. or .false.. When not present, .true. is assumed. This keyword currently enables gpu_density , gpu_hamiltonian , gpu_forces , and elsi_elpa_gpu . These keywords can also be used individually to turn on GPU acceleration in a specific part of FHI-aims. The ELPA eigensolver doesn't support HIP yet and this keyword should not be used for calculations that use both HIP and ELPA. Use gpu_density , gpu_hamiltonian and gpu_forces instead.

Tag: gpu_density (control.in)

Usage: gpu_density flag

Purpose: Use GPU acceleration when updating the charge density via density matrices. flag is optional. It can be either .true. or .false.. When not present, .true. is assumed. This keyword does nothing when using orbital-based density update.

Tag: gpu_hamiltonian (control.in)

Usage: gpu_hamiltonian flag

Purpose: Use GPU acceleration when integrating the Hamiltonian matrix. flag is optional. It can be either .true. or .false.. When not present, .true. is assumed.

Tag: gpu_forces (control.in)

Usage: gpu_forces flag

Purpose: Use GPU acceleration when calculating the Pulay forces and analytical stress tensor. flag is optional. It can be either .true. or .false.. When not present, .true. is assumed.

GPU aceleration of the ELPA eigensolver is controlled by elsi_elpa_gpu . See also Section 3.9. This keyword is not yet supported by HIP.

One important keyword when running GPU-accelerated calculations is points_in_batch , which sets the targeted number of points per batch. This parameter is a trade-off: increasing the number of points per batch increases the work done by the GPU per batch, increasing the efficiency of the GPU, but it also increases the number of basis elements interacting with a batch, increasing the memory usage. Due to technical details, some of this additional work is unnecessary, as it does not appreciably add to the integrals being evaluated.

The default value for `points_in_batch` based on early CPU-only benchmarks was set to 100. We have found that increasing this value to 200 is a better choice for our test architecture (Kepler Tesla GPUs) when using GPU acceleration, and we have set 200 as the default value when running any GPU-accelerating any tasks involving the batch integration scheme. The user should also play around with this parameter for their own architecture, particularly if they are using a different GPU architecture.

All GPU keywords may be set independently. We have found that the charge density update shows a significantly higher GPU-accelerated speed-up than the Hamiltonian integration (c.f. Figure G.1). If the user's architecture uses fast CPUs but slow GPUs, enabling GPU acceleration may actually slow down the calculation.

G.4.1 Memory Usage with GPU Acceleration

One hypothetical limitation in the current implementation of GPU acceleration in FHI-aims is GPU memory usage. All MPI ranks are assigned to one of the available GPUs, implying that a GPU will generally have more than one MPI rank assigned to it. All MPI ranks will offload their work during the compute-intensive cuBLAS call onto the assigned GPU. This creates two bottlenecks: not only do MPI ranks need to "wait their turn" behind other MPI ranks before the GPU processes their current batch, but each MPI rank will take up a portion of the GPU's memory. If a calculation runs out of memory when using GPU acceleration, some possible solutions are:

- Read Section 3.38, "Large-scale, massively parallel: Memory use, sparsity, communication, etc." of the manual. In particular, consider setting `use_local_index` to `.true.` . While there will be a time cost associated with enabling this keyword, the memory savings can be considerable.
- Use less MPI ranks per node. By having less MPI ranks per node, less MPI ranks will be bound to the GPUs on each node, reducing the overall GPU memory usage.

Summary

- Use keywords in `control.in` to enable GPU acceleration.
- Optimize `points_in_batch` for the architecture used.
- Test each GPU acceleration keyword individually to make sure there is a speed up compared to the CPU-only version for the architecture used.
- Try `use_local_index .true.` if your calculation runs out of memory. (This is true for all calculations, not just GPU-accelerated calculations.)

Appendix H

More on CMake

H.1 The build process

CMake itself is usually obtained from an official repository of a Linux distribution or built from source. When building from source, the only prerequisite is a C++ compiler.

The build process with CMake consists of two steps: the configuration stage, which generates the platform's native build files, and the compilation stage. (This is vaguely similar to the standard `configure` and `make` steps associated with Autotools.)

The configuration stage creates a set of persistent variables, which are contained in a file called CMakeCache.txt in the build directory. These are referred to as cache variables and they are the user-configurable settings for the project. All the important decisions such as which compiler to use, which libraries to link against, etc., are stored as cache variables. There are three main ways to perform the configuration stage (or set the contents of the CMake cache):

- Running `cmake` and specifying all the configuration variables on the command line. For example, running

  ```
  cmake -D CMAKE_Fortran_COMPILER=mpif90 ~aims
  ```

  from within the build directory, where `~aims` is the root source directory, sets mpif90 as the Fortran compiler. This approach is good for debugging and quick testing purposes but if a large number of variables is to be specified, it might not be very convenient. Also, no logic can be included this way.

- A better way is to keep the configuration variables in a separate file, an example of which is `initial_cache.example.cmake` in the root source directory of FHI-aims. It contains example initial values for some configuration variables, which you can edit to reflect your environment. Make a copy of it first, for example `initial_cache.cmake`. When done editing, run

  ```
  cmake -C ~aims/initial_cache.cmake ~aims
  ```

from the build directory. The `-C` option tells CMake to load a script, in this case `initial_cache.cmake`, which populates the initial cache. The loaded entries take priority over the project's default values. **A change of the content of the initial cache file has no effect after the first configuring.** Instead, it would be necessary to empty the build directory before using the cache file again. Alternatively, one could use a cache editor like `ccmake` to modify the configuration variables without emptying the whole build directory, as explained next.

- Run `cmake` first and then configure using a CMake GUI. In the case of FHI-aims, a bare `cmake` run is disallowed, since specifying at the least the Fortran compiler is required. In general, this is not a requirement for CMake projects. Two commonly used graphical front ends (or cache editors) are the `curses` based `ccmake` and the Qt-based `cmake-gui`. When using `ccmake`, issue

  ```
  cmake -D CMAKE_Fortran_COMPILER=mpif90 ~aims
  ccmake .
  ```

  from the build directory. Some CMake variables and options appear with a short help text to each variable displayed at the bottom in a status bar. Pressing 't' reveals all options. When done editing, press 'c' to reconfigure and 'g' to generate the native build scripts (e.g., makefiles). Pay attention when `ccmake` warns you that the cache variables have been reset. This will happen, for example, when changing the compiler, and will necessitate the reconfiguring of some variables. After configuring, it is a good idea to go through all the variables once more to check that everything is correct. Using `cmake-gui` is similar to `ccmake`:

  ```
  cmake -D CMAKE_Fortran_COMPILER=mpif90 ~aims
  cmake-gui .
  ```

  Besides a graphical window that opens, the usage of `cmake-gui` is analogous to `ccmake`.

When done configuring, FHI-aims can be compiled by issuing a compile command depending on the CMake generator used. A CMake generator is the function that writes the build files. The default behavior is to generate makefiles for Make, in which case the compile command is `make`. When using another generator such as Ninja (which coincides with the name of the actual build system, Ninja), the compile command is `ninja`. See `cmake -help` for which generators are available. It is actually unnecessary to think about generators at all when you run

```
cmake --build .
```

which is a generic command that always uses the correct compile command. When done compiling, an executable called `aims<...>` is produced in the root build directory. The default base name of the final target is `aims` and the `<...>` part depends on details such as the FHI-aims version number. An executable is not the only supported target. For instance, when building a shared library in a Linux environment, a library called `libaims<...>.so` is produced instead.

Note that in general, CMake can also run from the source directory, but this is not allowed in FHI-aims due to certain conflicts. It is a good practice anyway to compile in a separate build directory in order to support multiple build configurations simultaneously and to keep the source directory clean.

H.2 All CMake variables

The following variables and options should be sufficient to build a well optimized FHI-aims binary. In order to see all the CMake cache variables that the user has control over, open a CMake GUI and toggle the *advanced* mode (many of those will have little or no effect).

Attention! Take special care when setting the Fortran compiler, the compiler flags, and any libraries, especially the linear algebra libraries. A nonoptimal choice here could easily cost you a significant amount of computer time.

- `CMAKE_Fortran_COMPILER` — Name of the Fortran compiler executable. Use a full path if location not automatically detected.
- `CMAKE_Fortran_FLAGS` — Compilation flags that control the optimization level and other features that the compiler will use.
- `CMAKE_EXE_LINKER_FLAGS` — These flags will be used by the linker when creating an executable (i.e. equivalent to LDFLAGS)
- `LIB_PATHS` — List of directories to search in when linking against external libraries (e.g., "/opt/intel/mkl/lib/intel64")
- `LIBS` — List of libraries to link against (e.g., "mkl_blacs_intelmpi_lp64 mkl_scalapack_lp64")
- `INC_PATHS` — Additional directories containing header files.
- `USE_MPI` — Whether to use MPI parallelization when building FHI-aims. This should always be enabled except for rare debugging purposes. (Default: automatically determined by the compiler)
- `USE_SCALAPACK` — Whether to use Scalapack's parallel linear algebra subroutines and the basic linear algebra communications (BLACS) subroutines. It is recommended to always use this option. In particular, large production runs are not possible without it. The Scalapack libraries themselves should be set in `LIB_PATHS` and `LIBS`. (Default: automatically determined by `LIBS`)
- `ARCHITECTURE` — Can have multiple meanings, including specific handling of a few compilers' quirks (the PGI compiler, for example, needs a different call to `erf()`) and potentially optimization levels for CPU-specific extensions such as AVX. For many purposes, leaving this variable empty (using the so-called generic architecture) is good enough but do take the time to look into CPU-specific optimizations if you intend to run very large, demanding calculations.

- `BUILD_STATIC_LIBS` — Whether to build FHI-aims as a static library instead of as an executable. This propagates to subprojects like ELSI unless overridden. (Default: OFF, i.e. an executable)
- `BUILD_SHARED_LIBS` — Whether to build FHI-aims as a shared library instead of as an executable. This propagates to subprojects like ELSI unless overridden. (Default: OFF, i.e. an executable)
- `CMAKE_BUILD_TYPE` — If set to "Release", any flags defined in `CMAKE_Fortran_FLAGS_RELEASE` are appended to `CMAKE_Fortran_FLAGS`. If set to "Debug", any flags defined in `CMAKE_Fortran_FLAGS_DEBUG` are appended instead.
- `CMAKE_C_COMPILER` — C compiler.
- `CMAKE_C_FLAGS` — C compiler flags.
- `USE_CXX_FILES` — Whether source files written in C++ should be compiled into FHI-aims. (Default: OFF)
- `CMAKE_CXX_COMPILER` — C++ compiler.
- `CMAKE_CXX_FLAGS` — C++ compiler flags.
- `CMAKE_ASM_COMPILER` — Assembler. Usually not needed.
- `EXTERNAL_ELSI_PATH` — Path to the external ELSI installation. If empty, internal ELSI is used instead. See also external ELSI notes below. FHI-aims requires at least ELSI v2.8.2.
- `ENABLE_PEXSI` — Enable the PEXSI (pole expansion and selected inversion) density matrix solver. C and CXX compilers are mandatory for this purpose. PEXSI relies on the SuperLU_DIST and PT-SCOTCH libraries. By default, redistributed versions will be used. This option is ignored when an external ELSI installation is in use. (Default: OFF)
- `ENABLE_EIGENEXA` — Enable the EigenExa eigensolver. Requires an externally compiled EigenExa library. This option is ignored when an external ELSI installation is in use. (Default: OFF)
- `ENABLE_MAGMA` — Enable GPU-accelerated eigensolvers in the MAGMA library. Requires an externally compiled MAGMA library. This option is ignored when an external ELSI installation is in use. (Default: OFF)
- `ENABLE_SIPS` — Enable the SLEPc-SIPs eigensolver. Requires externally compiled SLEPc and PETSc libraries. This option is ignored when an external ELSI installation is in use. (Default: OFF)
- `ADD_UNDERSCORE` — In the redistributed PEXSI and SuperLU_DIST code (written in C and C++), there are calls to basic linear algebra routines such as `dgemm`. When `ADD_UNDERSCORE` is enabled, which is the default behavior, the C/C++

code will call `dgemm_` instead of `dgemm`. Therefore, turn this option off if you know routine names in your linear algebra library are not suffixed with an underscore. This option takes no effect if PEXSI is not enabled. It is ignored when an external ELSI installation is in use. (Default: ON)

- `ELPA2_KERNEL` — The ELPA eigensolver comes with a number of linear algebra "kernels" specifically optimized for some certain processor architectures. By default, FHI-aims uses a generic kernel which will compile with any Fortran compiler and will give reasonable speed. However, if one knows which specific computer chip one is using, it is possible to substitute this kernel with an architecture specific kernel and compile a faster version of ELPA. When using the internal version of ELSI/ELPA, available "kernels" other than the generic one may be selected by setting the `ELPA2_KERNEL` option to one of the following: "AVX" (Intel AVX), "AVX2" (Intel AVX2), and "AVX512" (Intel AVX512). This option is ignored when an external version of ELSI and/or ELPA is in use. (Default: a generic Fortran kernel will be used)

- `USE_EXTERNAL_ELPA` — Use an externally compiled ELPA library. Relevant libraries and include paths must be present in `LIBS`, `LIB_PATHS`, and `INC_PATHS`. This option is ignored when an external ELSI installation is in use. (Default: OFF)

- `USE_EXTERNAL_PEXSI` — When PEXSI is enabled, use an externally compiled PEXSI library. Relevant libraries and include paths must be present in `LIBS`, `LIB_PATHS`, and `INC_PATHS`. This option takes no effect if PEXSI is not enabled. It is ignored when an external ELSI installation is in use. (Default: OFF)

- `USE_CUDA` — Whether to use GPU acceleration in certain subroutines. See also appendix G. (Default: OFF)

  - `CMAKE_CUDA_COMPILER` — CUDA compiler. Automatically detected with CMake version ≥ 3.8.
  - `CMAKE_CUDA_FLAGS` — Flags for the CUDA compiler. Example: "-O3 -DAdd_ -arch=sm_70 -lcublas". With CMake version ≥ 3.18, it is recommended to define `CMAKE_CUDA_ARCHITECTURES`, which automatically sets "-arch=" etc.
  - `ENABLE_CUDA_BY_DEFAULT` — Automatically enable CUDA GPU acceleration at runtime (wherever possible). (Default: OFF)

- `USE_LIBXC` — Whether additional subroutines for exchange correlation functionals, provided in the LibXC library, should be used. It is advised to always use this. Please respect the open-source license of this tool and cite the authors if you use it. (Default: ON)

- `LIBXC_VERSION` — If given, this version of LibXC will be downloaded and compiled into FHI-aims. This variable is meant for developers for testing new features of LibXC, and must match exactly the LibXC version numbering (e.g. version.subversion.subsubversion). (Default: "", i.e. the shipped version is used)

- `USE_EXTERNAL_LIBXC` — Use an externally compiled LibXC library. Relevant libraries and include paths must be present in `LIBS`, `LIB_PATHS`, and `INC_PATHS`. (Default: OFF)
- `USE_CFFI` — Whether to provide a Python 2 or 3 interface to FHI-aims. (Default: OFF)
- `ELPA_MT` — Whether the hybrid MPI OpenMP version of ELPA is used. (Default: OFF)
- `USE_IPC` — Whether to support inter-process communication. (Default: OFF)
- `USE_iPI` — Whether to support path integral molecular dynamics through the i-PI python wrapper. (Default: ON)
- `USE_HDF5` — Whether to enable HDF5 support. If enabled, also set `INC_PATHS`, `LIB_PATHS`, and `LIBS` accordingly. (Default: OFF)
- `Fortran_MIN_FLAGS` — Minimal flags to be used instead of the primary flags to speed up compilation. (Default set by the current Fortran flags and the build type)
- `KNOWN_COMPILER_BUGS` — If set, reduced compiler flags are used instead of the primary flags. Options: "ifort-14.0.1-O3", "ifort-17.0.0-O3". (Default: none)
- `FFLAGS` — Flags used for compiling .f files. (Default: same as for .f90 files)
- `SERIAL_Fortran_COMPILER` — Deprecated. No effect.
- `USE_PLUMED` — Whether to use the PLUMED library. (Default: OFF)
- `USE_MPI_MODULE` — Use MPI module instead of "mpif.h" in Fortran code. (Default: OFF)

These variables do not affect performance in any way:

- `TARGET_NAME` — Base name for the primary target. Use this if you do not want the FHI-aims target name to be affected by things like the version number.
- `DEV` — Ignore the Fortran compiler check when running CMake in an empty build directory without initializing any cache variables.

External ELSI notes

- How to link to external ELSI built with external dependencies? For example, ELSI could have been built with external ELPA. In such a scenario, the FHI-aims variables `INC_PATHS`, `LIB_PATHS`, and `LIBS` need to point to the ELPA header files and libraries. If ELSI has no external dependencies, it is sufficient to only set `EXTERNAL_ELSI_PATH` in order to use that version of ELSI.

- When using CMake version less than 3.5.2, it is necessary to additionally point `LIB_PATHS` and `LIBS` to the external ELSI libraries. See `cmake/toolchains/ext_elsi.intel.cmake` as an example.
- FHI-aims requires at least ELSI version 2.8.2.

H.3 CMake for developers

The following is a very short overview of some of the more commonly used commands to give you a rough idea of the CMake syntax. This is, however, but not a substitute for a full tutorial, which you should work through yourself before contributing.

CMake support in FHI-aims is organized in files called CMakeLists.txt. There is one CMakeLists.txt in the root directory, which contains most of the functionality, and one in every subdirectory, containing primarily a list of source files.

In any CMake project, the first command in the topmost CMakeLists.txt is usually

```
cmake_minimum_required(VERSION x.x.x)
```

which sets the minimum required version of CMake for the project and ensures compatibility with that version or higher. This is followed by

```
project(MyProject VERSION 1.0.0 LANGUAGES C)
```

which sets the project name, version, and any languages used in the project (more languages can be enabled later). The basic syntax for setting a variable is `set(<var> <value>)`, like this:

```
set(LIBS mkl_sequential)
```

Variables can be referenced with the ${...} construct. For example,

```
set(LIBS ${LIBS} mkl_core)
message(${LIBS})
```

prints "mkl_sequential mkl_core" to the screen. This is a very basic usage of `set`, which can actually take several more arguments. The full functionality of the `set` or any other CMake command can be seen by either viewing the online CMake manual or using the `-help` argument to `cmake`:

```
cmake --help set
```

There is only one variable type in the CMake language, which is the string type. Even if some variables may be treated as numbers or booleans, they are still stored as strings.

Every statement is a command that takes a list of string arguments and has no return value. Thus, all CMake commands are of the form `command_name(arg1 ...)`. No command can be used directly as input for another command. Even control flow statements are commands:

```
if (USE_MPI)
  message("MPI parallelism enabled")
endif() # This is also a command. It takes no arguments.
```

A CMake-based buildsystem is organized as a set of high-level logical targets. Each target corresponds to an executable or library, or is a custom target containing custom commands. Dependencies between the targets are expressed in the buildsystem to determine the build order and the rules for regeneration in response to change. Executables and libraries are defined using the `add_executable` and `add_library` commands. Linking against libraries takes place via the `target_link_libraries` command:

```
add_library(mylib func_info.c mgga_c_scan.c xc_f03_lib_m.f90)
add_executable(myexe main.f90)
target_link_libraries(myexe mkl_intel_lp64 mkl_sequential mkl_core)
```

A library may be given by its full path, which is the standard practice, or by just the base name where the "-l" part is optional (both "`-lmkl_core`" and "`mkl_core`" are fine). In the latter case, directories to be linked against must be specified elsewhere. In addition to the standard locations, additional header directories may be specified using

```
include_directories(...)
```

`include_directories` accepts an additional `[AFTER|BEFORE]` argument, which determines whether to append or prepend to the current list of directories.

When do I need to reconfigure the build files? If a cache variable is modified, for example by using a cache editor, or if there are any other changes to the build settings, like when adding/removing a source file, the build files need to be regenerated. Fortunately, CMake can detect this and regenerate the build files automatically whenever the build command is issued. Thus, there is never a need to manually run `cmake` on the build directory except for the very first time. However, depending on the extent of your changes to the CMake scripts, the build configuration might sometimes have to be reset manually. Because CMake is incapable of tracking all the files that are generated during configuration, there is no `cmake clean` command. If you need to reset the build configuration, simply run `rm -rf` on the build directory.

Why is my compilation taking so long? CMake uses Make recursively when generating the build files, which has an adverse effect for projects with a large number of source files. There is no way around it except to switch to a different build system. When using a different generator (which generates build files for a different build system), the only change is in the initial CMake call. For example,

```
cmake -G Ninja -C initial_cache.cmake ~aims
```

chooses Ninja as the build system (otherwise the default is Make in Linux), initializes the cache using `initial_cache.cmake`, and specifies `~aims` as the source directory. The generator cannot be changed without emptying the build directory first. For users, it does not make a big difference which generator is used, but for developers it is advisable to use Ninja instead of Make as it is faster, especially for small incremental builds.

Appendix I

Building FHI-aims with a make.sys

This section contains a quick and practical explanation of the main steps using a `make.sys` file. This how FHI-aims used to be compiled in the past and is here for legacy reasons. Building via cmake is now the firmly recommended first choice and it is not guaranteed that building via `make.sys` will still work.

1. In the *src* directory, create a file called `make.sys` and open it with a text editor. Make sure you did *not* edit the file called `Makefile` as provided with the original distribution of FHI-aims if you choose to use and edit the `make.sys` file (which is recommended).

2. In order to build FHI-aims, you will need to inform the computer about which particular compilers, libraries, optimization flags and possible optional parts of the build process you intend to use. This is the purpose of `make.sys`. We here only cover a few most important keywords (variables) to be included in `make.sys`. Many more are available, often documented in the actual `Makefile` or, if nothing else, in the more detailed `Makefile.backend`, which controls the detailed pieces of the build process. Note that the syntax, particularly the spaces around the " = " signs, in `make.sys` are important since this file will be included in the `Makefile` and will have to be read by the `make` command further below.

3. The following is what a typical `make.sys` file could look like (see the https://aims-git.rz-berlin.mpg.de wiki for other examples for specific platforms). The explanation of all keywords follows below. Note that this is the copy of `make.sys` on the author's (VB's) laptop. You will need to edit every single variable – the directories to be used on other computers *will* be different. Blind copying and hoping for the best will not work.

```
FC = ifort
FFLAGS = -O3 -ip -fp-model precise -module $(MODDIR)
FMINFLAGS = -O0  -fp-model precise -module $(MODDIR)
F90MINFLAGS = -O0  -fp-model precise -module $(MODDIR)
F90FLAGS = $(FFLAGS)
ARCHITECTURE = Generic
LAPACKBLAS = -L/opt/intel/mkl/lib -I/opt/intel/mkl/include \
             -lmkl_intel_lp64 -lmkl_sequential -lmkl_core
```

```
USE_MPI = yes
MPIFC = mpif90
SCALAPACK = /usr/local/scalapack-2.0.2/libscalapack.a
CC = gcc
CCFLAGS =
USE_LIBXC = yes
```

4. Here is a list of each of these keywords' meanings:

   - `FC` : The name of the Fortran compiler you intend to use. This choice is not unimportant. On x86 platforms, Intel Fortran usually produces fast code, whereas other compilers (unfortunately, particularly free compilers such as gfortran) can lead to significantly slower (factor 2-3) FHI-aims runs later.

   - `FFLAGS` : These are compile-time and linker flags that control the optimization level that the compiler will use. Finding out which optimization level is fastest is worth your time, but note that real-world compilers can have bugs. In the worst case, this can mean numerically wrong results, something you should definitely care about. One way to test the broader correctness of a given FHI-aims build (later) is to run FHI-aims' regression tests on the computer you intend to use and make sure that all results are marked as correct. For example, for Intel Fortran, `-fp-model precise` is highly recommended. Unfortunately, we have no way to foresee all possible compiler bugs across all future platforms and compilers – testing is best. Please ask if needed (see Sec. 1.7 for where to find help).

   - `FMINFLAGS` specifies a lower optimization level for some subroutines that do not need optimization. `read_control.f90`, the subroutine that reads one of FHI-aims' main input files, is one such file that does not need high levels of optimization but could take very long to compile if a high optimization level were requested for it.

   - `F90MINFLAGS` and `F90FLAGS` are usually just copies of `FMINFLAGS` and `FFLAGS`, except for the few compilers (IBM's xlf) that might treat Fortran .f90 and (legacy) .f files differently.

   - `ARCHITECTURE` can have multiple meanings, including specific handling of a few compilers' quirks (the pgi compiler, for example, needs a different call to erf()) and potentially optimization levels for CPU-specific extensions (e.g., AVX - this can be worthwhile). For many purposes, "Generic" is good enough but do take the time to look into CPU-specific optimizations if you intend to run very large, demanding calculations.

   - `LAPACKBLAS` specifies the locations of numerical linear algebra subroutines, particularly the Basic Linear Algebra Subroutines (BLAS) and the higher-level Lapack subroutines. The location and names of these libraries will vary from computer to computer, but it is **VERY** important to select well-performing BLAS subroutines for a given computer – the effect on performance will be drastic. An additional item to ensure is that these BLAS libraries should **NEVER** try to use any internal

multithreading (for example, the `mkl_sequential` library quoted above is inherently single-threaded, which is normally what we want). FHI-aims is already very efficiently parallized for multiple processors. Requesting (say) 16 threads for each of (say) 16 parallel tasks on a parallel computer with 16 physical CPU cores would have the effect of trying to balance 256 threads within the computer, typically slowing execution down to a crawl. With FHI-aims, only ever use only a single thread per parallel task unless you have a special reason and know exactly what you are doing.

- `USE_MPI` will make sure that the code knows and will use the process-based Message Passing Interface (MPI) parallelization, which makes sure that FHI-aims can run in parallel both inside a single compute node as well as across a large number of nodes. In later production runs and unless you have a good reason not to do so, always use as many MPI tasks as there are physical processor cores available (no more, no less).

- `MPIFC` is the name of the wrapper command that ensures a correct compilation with a given Fortran compiler and a given MPI library. This command (often called `mpif90`) is also specific to a given computer system and to the installed MPI library.

- `SCALAPACK` specifies the location of the library that contains scalapack's parallel linear algebra subroutines and the so-called basic linear algebra communications (BLACS) subroutines. The author (VB) built his own version of this libary, but usually these subroutines are also supplied with standard linear algebra libraries such as Intel's Math Kernel Library (mkl).

- `CC` is the C compiler to be used.

- `CCFLAGS` could house any compiler flags needed for the C compiler. It is not worth doing this for performance reasons (very little impact) but some compilers may need other special instructions to work with Fortran.

- `USE_LIBXC` decides whether additional subroutines for exchange correlation functionals, provided in the `libxc` library, should be used. We are very much indebted to the authors of this library. Please respect their open-source license and cite them if you use their tools.

5. *Phew.* That was a lot of keywords. But this is computational science, and having a reasonable command of these pieces is worth our while. If you did figure them all out, close the `make.sys` file and continue to ...

6. ... build the code by typing `make -j scalapack.mpi`.

7. Do not despair. If the process above worked well, proceed to try a testrun and then, if you are up for it, the regression tests. If you received an error message during the build (that may well be the case), do not despair – try again and, if needed, seek help. This process is ultimately not rocket science and only a finite amount of pieces are needed. Seek help through one of the channels mentioned in Sec. 1.7 if needed.

8. There are other pieces that can help improve a build on a specific platform. For example, it can be quite desirable to build and link instead to a separate (standalone build) of the ELPA library (high-performance eigenvalue solver) and of the ELSI electronic structure infrastructure. For time and space reasons, this is not covered here presently, but it's worth investigating these libraries.

In general, the https://aims-git.rz-berlin.mpg.de Wiki is the appropriate place to look for detailed compiler settings for specific platforms. If you have a successful 'make.sys' file for your own setup, please add it there. The information given in this section is essential as it explains the process, but the platform specific remarks in the Wiki may help you save some time.

I.1 A more measured approach to building FHI-aims

This is a slower and step by step explanation of the build process, slightly different and somewhat redundant with Sec. I. Ultimately, your build process should ideally look somewhat like what is covered in Sec. I, and in particular, never edit the `Makefile` if you already have a file `make.sys` around. What follows is based on direct editing of the `Makefile` and should only be needed for practice purposes.

Starting from the end of Sec. 1.2 and 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".[1] 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>
```

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).

I.1.1 Cross-Compiling with a C Compiler

As noted in Section I, FHI-aims does provide some functionality that can only be accessed by compiling part of the code base with a C compiler, in some cases simply because a system call in question is not available from a Fortran interface.

Cross-compiling with a C compiler can be simple *if* your Fortran and C compiler use compatible interfaces. (This may not be always the case.) In the simplest case, adding the following variable to your `make.sys`:

```
CC = gcc
```

The CC variable should specify a C compiler compatible with your Fortran compiler on your specific computer system. Please make sure that the right C compiler is chosen for your system.

An additional variable `CCFLAGS` can be set to specify C compiler flags along with the `CC` variable above and will add C compiler flags that might be necessary for your particular computer setup.

[1] 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.)

I.2 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.

We also note that not all of these builds are routinely tested. At this time, it is not certain that all of them will still work out of the box. The information is kept here to make sure it is available. Please ask (see Section 1.7) if you encounter problems.

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 3.22 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 (at present, this is already specified in `plumed.inc`). 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.ipi` : Allows to interface with the i-PI path integral molecular dynamics wrapper[45]. 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.

- `Makefile.amd64_SSE` : This Makefile contains an example how to use FHI-aims together with a version of the ELPA library which is especially tuned for SSE vector instructions. Note that the GNU C compiler must be installed in order to produce running code from the specific assembler files. As the name "amd64" indicates this optimization is only reasonable on processors which support the amd64 instruction set. Note that "ELPA_ST" is set, to use the single-threaded version of ELPA.

- `Makefile.amd64_SSE_mt` : The same as Makefile.amd64_SSE, but by setting "ELPA_MT" the hybrid MPI OpenMP version of ELPA is used.

- `Makefile.amd64_AVX` : This Makefile contains an example how to use FHI-aims together with a version of the ELPA library which is especially tuned for AVX vector instructions. Note that the GNU C and C++ compiler must be installed in order to produce running code from the specific ELPA files which contain gcc intrinsic functions. As the name "amd64" indicates this optimization is only reasonable on processors which support the amd64 instruction set and which already "understand" AVX, i.e. Intel Sandybridge or newer. Note that "ELPA_ST" is set, to use the single-threaded version of ELPA.

- `Makefile.amd64_AVX_mt` : The same as Makefile.amd64_AVX, but by setting "ELPA_MT" the hybrid MPI OpenMP version of ELPA is used.

- `Makefile.hdf5` : The HDF5 module provides functions and routines to efficently make use of parallel writing and reading of data (mpi-io). The usual compiler options have to be set in the same way as in the normal Makefie. Additionally the path to your installation of hdf5 must be provided. If the environment variable *HDF5_HOME* is already set on your system nothing else has to be changed otherwise it has to be set in *Makefile.hdf5*.

Bibliography

[1] C. Adamo and V. Barone. *J. Chem. Phys.*, 110:6158, 1999. 66

[2] C. Van Alsenoy. *J. Comput. Chem.*, 9:620, 1988. 69 , 257

[3] C. Ambrosch-Draxl and J. O. Sofo. Linear optical properties of solids within the full-potential linearized augmented planewave method. *Computer Physics Communications*, (175):1–14, 2006. 363

[4] Alberto Ambrosetti, Anthony M. Reilly, Robert A. DiStasio, Jr., and Alexandre Tkatchenko. Long-range correlation energy calculated from coupled atomic response functions. *The Journal of Chemical Physics*, 140:18A508, 2014. 243

[5] H.C. Andersen. *J. Chem. Phys.*, 72:2384, 1980. 184

[6] Oliviero Andreussi, Ismaila Dabo, and Nicola Marzari. Revised self-consistent continuum solvation in electronic-structure calculations. *J. Chem. Phys.*, 136(6):064102–1–064102–20, 2012. 210 , 211 , 215 , 227

[7] V. I. Anisimov, editor. *Strong Coulomb correlations in electronic structure calculations*. Gordon and Breach, New York, 2000. 56

[8] V.I. Anisimov, J. Zaanen, and O. K. Andersen. Band theory and Mott insulators: Hubbard U instead of Stoner I. *Phys. Rev. B*, 44:943–954, Jul 1991. 232

[9] R. Armiento and Ann E. Mattsson. *Phys. Rev. B*, **72**:085108, 2005. 64

[10] 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. 536 , 539

[11] G. Aubert. An alternative to Wigner d-matrices for rotating real spherical harmonics. *AIP Adv.*, 3(6):062121, June 2013. 506

[12] N. Auer, L. Einkemmer, P. Kandolf, and A. Ostermann. Magnus integrators on multicore cpus and gpus. *Computer Physics Communications*, 228, 2018. 298 , 299

[13] A. Bagrets. Spin-polarized electron transport across metal-organic molecules: a density functional theory approach. *J. Chem. Theory Comput.*, 9:2801, 2013. 536

[14] J. Baker, J. Andzelm, A. Scheiner, and B. Delley. *J. Chem. Phys.*, 101:8894, 1994. 95

[15] A. Barducci, G. Bussi, and M. Parrinello. Well-tempered metadynamics: A smoothly converging and tunable free-energy method. *Phys. Rev. Lett.*, 100:020603, 2008. 528

[16] Albert P Bartók and Jonathan R Yates. Regularized scan functional. *The Journal of chemical physics*, 150(16):161101, 2019. 67

[17] C. I. Bayly, P. Cieplak, W. D. Cornell, and P. A. Kollman. A well-behaved electrostatic potential based method using charge restraints for deriving atomic charges: The resp model. *J. Phys. Chem.*, 97:10269–10280, 1993. 374

[18] A. D. Becke and K. E. Edgecombe. *J. Chem. Phys.*, 92:5397, 1990. 440

[19] AD Becke. On the large-gradient behavior of the density functional exchange energy. *J. Chem. Phys.*, 85:7184, 1986. 64 , 66

[20] A.D. Becke. *J. Chem. Phys.*, 88:1053, 1988. 64

[21] A.D. Becke and E.R. Johnson. Exchange-hole dipole moment and the dispersion interaction revisited. *J. Chem. Phys.*, 127:154108, 2007. 247

[22] J. Behler, B. Delley, S. Lorenz, K. Reuter, and M. Scheffler. *Phys. Rev. Lett.*, **94**:036104, 2005. 194

[23] J. Behler, B. Delley, K. Reuter, and M. Scheffler. *Phys. Rev. B*, **75**:115409, 2007. 194

[24] Paul Bendt and Alex Zunger. Simultaneous relaxation of nuclear geometries and electric charge densities in electronic structure theories. *Physical Review Letters*, 50(21):1684–1688, 1983. 170

[25] B. H. Besler, K. M. Merz, and P. A. Kollman. Atomic charges derived from semiempirical methods. *J. Comput. Chem.*, 11:431–439, 1990. 374

[26] Miguel A. Blanco, M. Flórez, and M. Bermejo. Evaluation of the rotation matrices in the basis of real spherical harmonics. *Comp. Theor. Chem.*, 419(1–3):19–27, December 1997. 506

[27] Miguel A. Blanco, M. Flórez, and M. Bermejo. Evaluation of the rotation matrices in the basis of real spherical harmonics. *Journal of Molecular Structure: THEOCHEM*, 419(1–3):19 – 27, 1997. 415

[28] 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. 33 , 39 , 70 , 86 , 89 , 98 , 99 , 100 , 110 , 111 , 116 , 132 , 134 , 139 , 144 , 147 , 148 , 154 , 178 , 315 , 342 , 364 , 446 , 461 , 464 , 476

[29] 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. 178

[30] A. Bondi. van der waals volumes and radii. *J. Phys. Chem.*, 68:441–451, 1964. 374

[31] Silvana Botti and Matteo Gatti. *Fundamentals of Time-Dependent Density Functional Theory*, chapter The Microscopic Description of a Macroscopic Experiment, pages 29–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. 366

[32] S.F. Boys and I. Shavitt. *University of Wisconsin Rept.*, WIS-AF-13, 1959. 69 , 257

[33] C. J. Bradley and A. P. Cracknell. *The mathematical theory of symmetry in solids : representation theory for point groups and space groups*. Clarendon Press, 1972. 415

[34] C. M. Breneman and K. B. Wiberg. Determining atom-centered monopoles from molecular electrostatic potentials. the need for high sampling density in formamide conformational analysis. *J. Comput. Chem.*, 11:361–373, 1990. 374

[35] Kurt R. Brorsen, Yang Yang, and Sharon Hammes-Schiffer. Multicomponent Density Functional Theory: Impact of Nuclear Quantum Effects on Proton Affinities and Geometries. *J. Phys. Chem. Lett.*, 8:3488–3493, 2017. 315

[36] Adam Bruner, Daniel LaMaster, and Kenneth Lopata. Accelerated broadband spectra using transition dipole decomposition and padé approximants. *Journal of Chemical Theory and Computation*, 12(8), 2016. 293

[37] P. Bultinck, C. Van Alsenoy, P.W. Ayers, and R. Carbo-Dorca. Critical analysis and extension of the hirshfeld atoms in molecules. *J. Chem. Phys.*, **126**:144111, 2007. 449 , 450

[38] G. Bussi, D. Donadio, and M. Parrinello. Canonical sampling through velocity rescaling. *J. Chem. Phys.*, 126:014101, 2007. 185

[39] C. Campana, B. Mussard, and T. K. Woo. Electrostatic potential derived atomic charges for periodic systems using a modified error functional. *J. Chem. Theory Comput.*, 5:2866–2878, 2009. 374 , 375

[40] Mark E. Casida. Time-depenent density-functional response theory for molecules. *Theoretical and Computational Modeling of NLO and Electronic Materials*, 1996. 286

[41] Alberto Castro, Miguel A. L. Marques, and Angel Rubio. Propagators for the Time-dependent Kohn-Sham Equations. *The Journal of Chemical Physics*, 121(8):3425–3433, 2004. 298

[42] D. M. Ceperley and B. J. Alder. Ground state of the electron gas by a stochastic method. *Phys. Rev. Lett.*, **45**:566–569, 1980. 64

[43] 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. 185 , 186

[44] Michele Ceriotti, Giovanni Bussi, and Michele Parrinello. Colored-noise thermostats à la carte. *Journal of Chemical Theory and Computation*, 6(4):1170–1180, 2010. 185 , 186

[45] Michele Ceriotti, Joshua More, Michele, and David Manolopoulos. i-pi: A python interface for ab initio path integral molecular dynamics simulations. *Comp. Phys. Comm.*, 185:1019–1026, 2014. 187 , 596

[46] Michele Ceriotti, Michele Parrinello, Thomas E. Markland, and David E. Manolopoulos. Efficient stochastic thermostatting of path integral molecular dynamics. *The Journal of Chemical Physics*, 133(12):124104, 2010. 185 , 186

[47] D.J. Chadi and M.L. Cohen. *Phys. Rev. B*, **8**:5747, 1973. 77

[48] Che Ting Chan, Klaus Peter Bohnen, and KM Ho. Accelerating the convergence of force calculations in electronic-structure computations. *Physical Review B*, 47(8):4771–4774, 1993. 170

[49] D.-J. Chen, A. C. Stern, B. Space, and J. K. Johnson. Atomic charges derived from electrostatic potential for molecular and periodic systems. *J. Phys. Chem. A*, 114:10225–10233, 2010. 374 , 375

[50] L. E. Chirlian and M. M. Francl. Atomic charges derived from electrostatic potentials: A detailed study. *J. Comput. Chem.*, 8:894–905, 1987. 374

[51] L. A. Constantin, E. Fabiano, and F. Della Sala. *Phys. Rev. B*, 86:035130, 2012. 65

[52] S. R. Cox and D. E. Williams. Representation of the molecular electrostatic potential by a net atomic charge model. *J. Comput. Chem.*, 2:304–323, 1981. 374

[53] Christopher J. Cramer and Donald G. Truhlar. A universal approach to solvation modeling. *Acc. Chem. Res.*, 41(6):760–768, 2008. 210

[54] G. Crooks. Nonequilibrium measurements of free energy differences for microscopically reversible markovian systems. *J. Stat. Phys.*, 90:1481, 1998. 528

[55] Stefano de Gironcoli. Lattice dynamics of metals from density-functional perturbation theory. *Phys. Rev. B*, 51:6773–6776, Mar 1995. 347

[56] B. Delley. *J. Chem. Phys.*, 92:508, 1990. 89

[57] B. Delley. *J. Comp Chem.*, 17:1152, 1995. 87

[58] Katharina Diller, Florian Klappenberger, Francesco Allegretti, Anthoula C. Papageorgiou, Sybille Fischer, David A. Duncan, Reinhard J. Maurer, Julian A. Lloyd, Seung Cheol Oh, Karsten Reuter, and Johannes V. Barth. Temperature-dependent templated growth of porphine thin films on the (111) facets of copper and silver. *The Journal of Chemical Physics*, 141(14):144703, 2014. 200

[59] M. Dion, H. Rydberg, E. Schröder, D. C. Langreth, , and B. I. Lundqvist. *Phys. Rev. Lett.*, 92:246401, 2004. 62 , 68 , 240 , 249 , 250 , 254

[60] R. M. Dreizler and E. K. U. Gross. *Density Functional Theory*. Springer, Berlin, 1990. 9

[61] S. L. Dudarev, G. A. Botton, S. Y. Savrasov, C. J. Humphreys, and A. P. Sutton. Electron-energy-loss spectra and the structural stability of nickel oxide: An LSDA+U study. *Phys. Rev. B*, 57:1505–1509, Jan 1998. 233

[62] Brett I. Dunlap, Notker Rösch, and S. B. Trickey. Variational fitting methods for electronic structure calculations. *Mol. Phys.*, 108(21):3167, 2010. 271

[63] C Dupont, O Andreussi, and N Marzari. Self-consistent continuum solvation (SCCS): the case of charged systems. *J. Chem. Phys.*, 139(21):214110, December 2013. 211

[64] C. Eckart. *Phys. Rev.*, **47**:552, 1935. 167 , 170

[65] K. Eichkorn, O. Treutler, H. Öhm, M. Häser, and R. Ahlrichs. *Chem. Phys. Lett.*, 240:283, 1995. 69 , 257

[66] F. Evers and A. Arnold. Molecular conductance from ab initio calculations: Self energies and absorbing boundary conditions. *arXiv:cond-mat/0611401v1*, 2006. 539

[67] 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. 64

[68] J. L. Fattebert and F. Gygi. Density functional theory for efficient ab initio molecular dynamics simulations in solution. *J. Comput. Chem.*, 23(6):662–666, 2002. 227

[69] Jakob Filser, Karsten Reuter, and Harald Oberhofer. Piecewise multipole-expansion implicit solvation for arbitrarily shaped molecular solutes. *Journal of Chemical Theory and Computation*, 18(1):461–478, 2022. 210 , 211 , 212 , 213 , 214 , 215 , 216

[70] D. Frenkel and B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, second edition, 2002. 178

[71] 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. 535

[72] C.-L. Fu and K.-M. Ho. *Phys. Rev. B*, **28**:5480, 1983. 132

[73] Liang Fu and C. L. Kane. Topological insulators with inversion symmetry. *Phys. Rev. B*, 76:045302, Jul 2007. 352

[74] Liang Fu, C. L. Kane, and E. J. Mele. Topological insulators in three dimensions. *Phys. Rev. Lett.*, 98:106803, Mar 2007. 351

[75] 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. 73 , 206 , 207 , 208

[76] L. Gallandi and T. Körzdörfer. *Journal of Chemical Theory and Computation*, 11:5391–5400, 2015. 66

[77] M.J. Gillan. *J. Phys.:Condens. Matter*, 1:689, 1989. 133

[78] S. Goedecker and K. Maschke. Transferability of pseudopotentials. *Physical Review A*, 45:88–93, 1992. 73

[79] D. Golze. Dataset in NOMAD repository: Example from "CORE65 benchmark set", 2020. https://nomad-lab.eu/prod/rae/gui/entry/id/tncxyMTBTGikrP6sgMPuQw/l5TVvWyUkF2KWnyQb3P1gwcMcm3W. 259

[80] D. Golze, M. Dvorak, and P. Rinke. The GW compendium: A practical guide to theoretical photoemission spectroscopy. *Front. Chem.*, 7:377, Dec 2019. 59 , 258

[81] D. Golze, L. Keller, and P. Rinke. Accurate Absolute and Relative Core-Level Binding Energies from GW. *J. Phys. Chem. Lett.*, 11(5):1840–1847, 2020. 264 , 266

[82] D. Golze, J. Wilhelm, M. J. van Setten, and P. Rinke. Core-Level Binding Energies from GW: An Efficient Full-Frequency Approach within a Localized Basis. *J. Chem. Theory Comput.*, 14(9):4856–4869, 2018. 257 , 258 , 259 , 262

[83] Adrián Gómez Pueyo, Miguel A. L. Marques, Angel Rubio, and Alberto Castro. Propagators for the Time-Dependent Kohn-Sham Equations: Multistep, Runge-Kutta, Exponential Runge-Kutta, and Commutator Free Magnus Methods. *Journal of Chemical Theory and Computation*, 14(6):3040–3052, 2018. 298

[84] B Grabowski, L Ismer, T Hickel, and J Neugebauer. *Phys. Rev. B*, 79:134106, 2009. 190

[85] Dominik Gresch, Gabriel Autès, Oleg V. Yazyev, Matthias Troyer, David Vanderbilt, B. Andrei Bernevig, and Alexey A. Soluyanov. Z2pack: Numerical implementation of hybrid wannier centers for identifying topological materials. *Phys. Rev. B*, 95:075146, Feb 2017. 351

[86] S. Grimme. *J. Chem. Phys.*, **20**:9095, 2003. 61 , 68

[87] 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. 55

[88] P. Guetlein, L. Lang, K. Reuter, J. Blumberger, and H. Oberhofer. Toward first-principles-level polarization energies in force fields: A gaussian basis for the atom-condensed kohn–sham method. *The Journal of Chemical Theory and Computation*, 15:4516–4525, 2019. 427

[89] 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. 254

[90] G. Mills K. W. Jacobsen H. Jonsson. *Nudged Elastic Band Method for Finding Minimum Energy Paths of Transitions*. World Scientific, 1998. 507

[91] A. J. S. Hamilton. Uncorrelated modes of the non-linear power spectrum. *Mon. Not. R. Astron. Soc.*, 312(2):257–284, 2000. 273

[92] Andrew J. S. Hamilton. Fftlog, 2000. 273

[93] B. Hammer, L.B. Hansen, and J.K. Nørskov. *Phys. Rev. B*, 59:7413, 1999. 64

[94] M. J. Han, T. Ozaki, and J. Yu. O(N) LDA+U electronic structure calculation method based on the nonorthogonal pseudoatomic orbital basis. *Phys. Rev. B*, 73:045110, Jan 2006. 233

[95] 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. 69

[96] V. Havu, V. Blum, P. Havu, and M. Scheffler. Efficient o(n) integration for all-electron electronic structure calculation using numeric basis functions. *J. Comput. Phys.*, **228**:8367, 2009. 86 , 87 , 88 , 91 , 465 , 574

[97] L. Hedin. *Phys. Rev.*, 139:A796, 1965. 68

[98] Lars Hedin. On correlation effects in electron spectroscopies and the GW approximation. *J. Phys.: Condens. Matter*, 11:R489, 1999. 263

[99] Joscha Hekele and Peter Kratzer. Real-time time-dependent density functional theory within FHI-aims. *arXiv:2008.08845*, 2020. 290

[100] 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. 516

[101] Graeme Henkelman and Hannes Jonsson. Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *Journal of Chemical Physics*, 113(22):9978–9985, 2000. 507

[102] 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. 179

[103] Jan Hermann. Libmbd. Code as git repository. 241 , 243

[104] Jan Hermann and Alexandre Tkatchenko. Density Functional Model for van der Waals Interactions: Unifying Many-Body Atomic Approaches with Nonlocal Functionals. *Physical Review Letters*, 124:146401, 2020. 244

[105] Jochen Heyd, Gustavo E. Scuseria, and Matthias Ernzerhof. *J. Chem. Phys.*, 118:8207, 2003. 65 , 66

[106] Jochen Heyd, Gustavo E. Scuseria, and Matthias Ernzerhof. *J. Chem. Phys.*, 124:219906, 2006. 65

[107] D. Hicks, M. J. Mehl, E. Gossett, C. Toher, O. Levy, R. M. Hanson, G. Hart, and S. Curtarolo. The AFLOW Library of Crystallographic Prototypes: Part 2. *ArXiv e-prints*, 2018. 161

[108] B. Himmetoglu, A. Floris, S. de Gironcoli, and M. Cococcioni. "Hubbard-corrected DFT energy functionals: The LDA+U description of correlated systems". *Int. J. Quantum Chem.*, 114(1):14–49, 2014. 232

[109] F.L. Hirshfeld. *Theor. Chim. Acta (Berl.)*, 44:129, 1977. 449

[110] P. Hohenberg and W. Kohn. *Phys. Rev. B*, 136:864, 1964. 9

[111] H. Hu, Z. Lu, and W. yang yang yang yang. Fitting molecular electrostatic potentials from quantum mechanical calculations. *J. Chem. Theory Comput.*, 3:1004–1013, 2007. 374

[112] J. Hubbard. Electron Correlations in Narrow Energy Bands. *Proc. R. Soc. A*, 276(1365):238–257, 1963. 232

[113] William P. Huhn and Volker Blum. One-hundred-three compound band-structure benchmark of post-self-consistent spin-orbit coupling treatments in density functional theory. *Phys. Rev. Materials*, 1:033803, 2017. 52 , 111 , 112 , 113 , 115 , 476

[114] Kerwin Hui and Jeng-Da Chai. Scan-based hybrid and double-hybrid density functionals from models without fitted parameters. *J. Chem. Phys.*, 144:044114, 2016. 67

[115] Felix Hummel, Theodoros Tsatsoulis, and Andreas Grüneis. Low rank factorization of the coulomb integrals for periodic coupled cluster theory. *The Journal of chemical physics*, 146(12):124105, 2017. 326

[116] Arvid Conrad Ihrig, Jürgen Wieferink, Igor Ying Zhang, Matti Ropo, Xinguo Ren, Patrick Rinke, Matthias Scheffler, and Volker Blum. Accurate localized resolution of identity approach for linear-scaling hybrid density functionals and for many-body perturbation theory. *New Journal of Physics*, 17(9):093020, 2015. 163 , 256 , 257 , 258 , 270 , 271 , 274 , 277

[117] H. Ishida, Y. Nagai, and A. Kidera. *Chem. Phys. Lett.*, 282(2):115, 1998. 184

[118] 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. 480

[119] Manoj K. Jana, Ruyi Song, Haoliang Liu, Dipak Rajkhanal, Svenja M. Janke, Chi Liu, Rundong Zhao, Z. Valy Vardeny, Volker Blum, and David B. Mitzi. Organic-to-inorganic structural chirality transfer in a 2d hybrid perovskite: Impact on rashba-dresselhaus spin-orbit coupling. *Nature Communications*, 11(1):4699, 2020. 117

[120] C. Jarzynski. Nonequilibrium equality for free energy differences. *Phys. Rev. Lett*, 78:2690, 1997. 528

[121] Stig Rune Jensen, Santanu Saha, José A. Flores-Livas, William Huhn, Volker Blum, Stefan Goedecker, and Luca Frediani. The elephant in the room of density functional theory calculations. *The Journal of Physical Chemistry Letters*, 8(7):1449–1457, 2017. PMID: 28291362. 70 , 73

[122] Weile Jia, Dong An, Lin-Wang Wang, and Lin Lin. Fast Real-time Time-dependent Density Functional Theory Calculations with the Parallel Transport Gauge. 2018. 299

[123] Jmol. An open-source java viewer for chemical structures in 3D. http://www.jmol.org/. 485 , 486

[124] E R Johnson. The exchange-hole dipole moment dispersion model. In A Otero-de-la-Roza and G A DiLabio, editors, *Non-covalent Interactions in Quantum Chemistry and Physics*, chapter 5, pages 169–194. Elsevier, 2017. 247

[125] J. Junquera, O. Paz, D. Sanchez-Portal, and E. Artacho. *Phys. Rev. B*, 64:235111, 2001. 81

[126] C. L. Kane and E. J. Mele. Z_2 topological order and the quantum spin hall effect. *Phys. Rev. Lett.*, 95:146802, Sep 2005. 350 , 351

[127] Levi Keller, Volker Blum, Patrick Rinke, and Dorothea Golze. Relativistic correction scheme for core-level binding energies from gw. *The Journal of Chemical Physics*, 153(11):114110, 2020. 264

[128] G.P. Kerker. *Phys. Rev. B*, **23**:3082, 1981. 147

[129] R. D. King-Smith and David Vanderbilt. Theory of polarization of crystalline solids. *Phys. Rev. B*, 47:1651–1654, Jan 1993. 349

[130] A. Klamt and G. Schüürmann. COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient. *J. Chem. Soc., Perkin Trans. 2*, (5):799–805, January 1993. 210

[131] L. Kleinman and D. M. Bylander. Efficacious form for model pseudopotentials. *Phys. Rev. Lett.*, 48:1425–1428, May 1982. 206

[132] Florian Knoop, Matthias Scheffler, Christian Carbogno, et al. Fhi-vibes: _ab initio_ vibrational simulations. *Journal of Open Source Software*, 5(56):2671, 2020. 497

[133] Franz Knuth, Christian Carbogno, Viktor Atalla, Volker Blum, and Matthias Scheffler. All-electron formalism for total energy strain derivatives and stress tensor components for numeric atom-centered orbitals. *Computer Physics Communications*, 190:33–50, 2015. 163 , 277

[134] W. Kohn and L.J. Sham. *Phys. Rev.*, 140:A1133, 1965. 9

[135] M. Kohout and A. Savin. *Int. J. Quantum Chem.*, 60:875, 1996. 440

[136] Jiří Kolafa. Numerical integration of equations of motion with a Self-Consistent field given by an implicit equation. *Mol. Simul.*, 18:193, 1996. 179

[137] 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. 132 , 133

[138] J. B. Krieger, Y. Li, and G. J. Iafrate. *Phys. Lett. A*, 146:256, 1990. 73 , 74

[139] J. B. Krieger, Y. Li, and G. J. Iafrate. *Int. J. Quantum. Chem.*, 41:489, 1992. 73 , 74

[140] J. B. Krieger, Y. Li, and G. J. Iafrate. *Phys. Rev. A*, 45:101, 1992. 73 , 74

[141] Aliaksandr V. Krukau, Oleg A. Vydrov, Artur F. Izmaylov, and Gustavo E. Scuseria. *J. Chem. Phys.*, 125:224106, 2006. 65 , 66

[142] 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. 178

[143] 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. 528

[144] T. Kunert and R. Schmidt. Non-adiabatic Quantum Molecular Dynamics: General Formalism and case Study H2+ in Strong Laser Fields. *The European Physical Journal D*, pages 15–24, 2003. 292

[145] A. Laio and M. Parrinello. Escaping free energy minima. *Proc. Natl. Acad. Sci.*, 20:12562, 2002. 528

[146] Ask Hjorth Larsen, Jens Jørgen Mortensen, Jakob Blomqvist, Ivano E Castelli, Rune Christensen, Marcin Dułak, Jesper Friis, Michael N Groves, Bjørk Hammer, Cory Hargus, et al. The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, 2017. 497

[147] V.I. Lebedev. *Zh. Vychisl. Mat. mat. Fiz.*, 15:48, 1975. 87

[148] V.I. Lebedev. *Zh. Vychisl. Mat. mat. Fiz.*, 16:293, 1976. 87

[149] V.I. Lebedev and D.N. Laikov. *Doklady Mathematics*, 59:477, 1999. 87

[150] C.L. Lee, W. Yang, and R.G. Parr. *Phys. Rev. B*, 37:785, 1988. 64

[151] Susi Lehtola, Conrad Steigemann, Micael J. T. Oliveira, and Miguel A. L. Marques. Recent developments in libxc—a comprehensive library of functionals for density functional theory. *SoftwareX*, 7:1–5, 2018. 67 , 287

[152] Kurt Lejaeghere, Gustav Bihlmayer, Torbjörn Björkman, Peter Blaha, Stefan Blügel, Volker Blum, Damien Caliste, Ivano E. Castelli, Stewart J. Clark, Andrea Dal Corso, Stefano de Gironcoli, Thierry Deutsch, John Kay Dewhurst, Igor Di Marco, Claudia Draxl, Marcin Dułak, Olle Eriksson, José A. Flores-Livas, Kevin F. Garrity, Luigi Genovese, Paolo Giannozzi, Matteo Giantomassi, Stefan Goedecker, Xavier Gonze, Oscar Grånäs, E. K. U. Gross, Andris Gulans, François Gygi, D. R. Hamann, Phil J. Hasnip, N. A. W. Holzwarth, Diana Iușan, Dominik B. Jochym, François Jollet, Daniel Jones, Georg Kresse, Klaus Koepernik, Emine Küçükbenli, Yaroslav O. Kvashnin, Inka L. M. Locht, Sven Lubeck, Martijn Marsman, Nicola Marzari, Ulrike Nitzsche, Lars Nordström, Taisuke Ozaki, Lorenzo Paulatto, Chris J. Pickard, Ward Poelmans, Matt I. J. Probert, Keith Refson, Manuel Richter, Gian-Marco Rignanese, Santanu Saha, Matthias Scheffler, Martin Schlipf, Karlheinz Schwarz, Sangeeta Sharma, Francesca Tavazza, Patrik Thunström, Alexandre Tkatchenko, Marc Torrent, David Vanderbilt, Michiel J. van Setten, Veronique Van Speybroeck, John M. Wills, Jonathan R. Yates, Guo-Xu Zhang, and Stefaan Cottenier. Reproducibility in density functional theory calculations of solids. *Science*, 351(6280), 2016. 10 , 70 , 112 , 476

[153] C. Lessig, T. de Witt, and E. Fiume. Efficient and accurate rotation of finite spherical harmonics expansions. *J. Comput. Phys.*, 231(2):243–250, January 2012. 506

[154] Sergey V. Levchenko, Xinguo Ren, Jürgen Wieferink, Rainer Johanni, Patrick Rinke, Volker Blum, and Matthias Scheffler. Hybrid functionals for large periodic systems in an all-electron, numeric atom-centered basis framework. *Computer Physics Communications*, 192:60 – 69, 2015. 256 , 271 , 277 , 279

[155] R. Lindh, A. Bernhardsson, G. Karlström, and P.-Å. Malmqvist. *Chem. Phys. Lett.*, 241:423, 1995. 161 , 164 , 172 , 175 , 471

[156] S. Lizzit, A. Baraldi, A. Groso, K. Reuter, M. V. Ganduglia-Pirovano, C. Stampfl, M. Scheffler, M. Stichler, C. Keller, W. Wurth, and D. Menzel. *Phys. Rev. B*, 63:205419, 2001. 200

[157] 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. 208

[158] G. Makov and M. C. Payne. *Phys. Rev. B*, 51:4014, 1995. 535

[159] M. Manninen, R. Nieminen, and P. Hautojärvi. *Phys. Rev. B*, 12:4012, 1975. 147

[160] M. Mantina, A. C. Chamberlin, R. Valero, C. J. Cramer, and D. G. Truhlar. Consistent van der waals radii for the whole main group. *J. Phys. Chem. A*, 113:5806–5812, 2009. 374

[161] Aleksandr V. Marenich, Christopher J. Cramer, and Donald G. Truhlar. Universal solvation model based on solute electron density and on a continuum model of the solvent defined by the bulk dielectric constant and atomic surface tensions. *J. Phys. Chem. B*, 113(18):6378–6396, 2009. 210

[162] Aleksandr V. Marenich, Christopher J. Cramer, and Donald G. Truhlar. Generalized born solvation model sm12. *J. Chem. Theory Comput.*, 9(1):609–620, 2013. 210

[163] Miguel Marques, Neepa Maitra, Fernando Nogueira, Eberhard Gross, and Angel Rubio. *Fundamentals of Time-Dependent Density Functional Theory*, volume 837. 01 2012. 290

[164] Michael J. Mehl, David Hicks, Cormac Toher, Ohad Levy, Robert M. Hanson, Gus Hart, and Stefano Curtarolo. The aflow library of crystallographic prototypes: Part 1. *Computational Materials Science*, 136:S1 – S828, 2017. 161

[165] B. Mennucci, J. Tomasi, R. Cammi, J. R. Cheeseman, M. J. Frisch, F. J. Devlin, S. Gabriel, and P. J. Stephens. Polarizable continuum model (pcm) calculations of solvent effects on optical rotations of chiral molecules. *J. Phys. Chem. A*, 106(25):6102–6113, 2002. 210

[166] N.D. Mermin. *Phys. Rev.*, 137:A1441, 1965. 132 , 190

[167] M. Methfessel and A. T. Paxton. *Phys. Rev. B*, **40**:3616, 1989. 132

[168] Evgeny Moerman, Felix Hummel, Andreas Grüneis, Andreas Irmler, and Matthias Scheffler. Interface to high-performance periodic coupled-cluster theory calculations with atom-centered, localized basis functions. *Journal of Open Source Software*, 7(74):4040, 2022. 326

[169] F. A. Momany. Determination of partial atomic charges from ab initio molecular electrostatic potentials. application to formamide, methanol, and formic acid. *J. Chem. Phys.*, 82:592–601, 1978. 374

[170] H.J. Monkhorst and J.D. Pack. *Phys. Rev. B*, **13**:5188, 1976. 76 , 478 , 479

[171] R.S. Mulliken. *J. Chem. Phys.*, 23:1833, 1955. 377 , 453

[172] D. Nabok, P. Puschnig, and C. Ambrosch-Draxl. *Phys. Rev. B*, 77:245316, 2008. 249

[173] Francesco Nattino, Cristina Díaz, Bret Jackson, and Geert-Jan Kroes. *Phys. Rev. Lett.*, 108(23):236104, 2012. 64

[174] R. Nieminen. *J. Phys. F*, 7:375, 1977. 147

[175] Anders M. N. Niklasson, C. J. Tymczak, and Matt Challacombe. Time-Reversible Born-Oppenheimer molecular dynamics. *Phys. Rev. Lett.*, 97:123001, 2006. 183

[176] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 2. edition, 2006. 175

[177] H. Oberhofer and J. Blumberger. Revisiting electronic couplings and incoherent hopping models for electron transport in crystalline c60 at ambient temperatures. *Phys. Chem. Chem. Phys*, 14:13846–13852, 2012. 404

[178] Ari Ojanperä, Ville Havu, Lauri Lehtovaara, and Martti Puska. Nonadiabatic Ehrenfest Molecular Dynamics within the Projector Augmented-wave Method. *The Journal of Chemical Physics*, 136(14):144103, 2012. 292

[179] Jochen Heyd Aliaksandr V. Krukau Oleg A. Vydrov and Gustavo E. Scuseria. *J. Chem. Phys.*, 125:074106, 2006. 66

[180] A. Otero-de-la-Roza and E. R. Johnson. Van der Waals interactions in solids using the exchange-hole dipole moment. *J. Chem. Phys.*, 136:174109, 2012. 247

[181] J. P. Perdew, K. Burke, and M. Ernzerhof. Generalized gradient approximation made simple. *Phys. Rev. Lett.*, **77**:3865–3868, 1997. 64 , 66

[182] J. P. Perdew, J. A. Chevary, S. H. Vosko, K. A. Jackson, M. R. Pederson, D. J. Singh, and C. Fiolhais. Atoms, molecules, solids, and surfaces: Applications of the generalized gradient approximation for exchange and correlation. *Phys. Rev. B*, **46**:6671–6687, 1992. 64

[183] J. P. Perdew and Y. Wang. Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B*, **45**:13244–13249, 1992. 64

[184] J. P. Perdew and A. Zunger. Self-interaction correction to density-functional approximations for many-electron systems. *Phys. Rev. B*, **23**:5048–5079, 1981. 64

[185] J.P. Perdew, A. Ruzsinszky, G.I. Csonka, L.A. Constantin, and J. Sun. Workhorse semilocal density functional for condensed matter physics and quantum chemistry. *Phys. Rev. Lett.*, 103:026403, 2009. 65

[186] J.P. Perdew, A. Ruzsinszky, G.I. Csonka, L.A. Constantin, and J. Sun. Erratum: Workhorse semilocal density functional for condensed matter physics and quantum chemistry. *Phys. Rev. Lett.*, 106:179902(E), 2011. 65

[187] 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. 64 , 66

[188] Baron Peters, Andreas Heyden, Alexis T. Bell, and Arup Chakraborty. A growing string method for determining transition states: Comparison to the nudged elastic band and string methods. *The Journal of Chemical Physics*, 120(17):7877–7886, 2004. 507

[189] 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. 58 , 232 , 235

[190] R. Peverati and D.G. Truhlar. Improving the accuracy of hybrid meta-gga density functionals by range separation. *J. Phys. Chem. Lett.*, 2(21):2810, 2011. 67

[191] R. Peverati and D.G. Truhlar. M11-l: A local density functional that provides improved accuracy for electronic structure calculations in chemistry and physics. *J. Phys. Chem. Lett.*, 3:117, 2011. 64

[192] Bernd G. Pfrommer, Michel Côté, Steven G. Louie, and Marvin L. Cohen. Relaxation of Crystals with the Quasi-Newton Method. *Journal of Computational Physics*, 131(1):233–240, 1997. 172 , 175

[193] P. Pulay. Convergence acceleration of iterative sequences. the case of scf iteration. *Chem. Phys. Lett.*, **73**:393–398, 1980. 144

[194] Peter Pulay and Géza Fogarasi. Fock matrix dynamics. *Chem. Phys. Lett.*, 386:272–278, 2004. 179

[195] P. Pyykkö. *Chem. Rev.*, 88:563, 1988. 476

[196] 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. 55 , 71 , 73 , 74 , 256 , 257 , 258

[197] X. Ren, P. Rinke, G. E. Scuseria, and M. Scheffler. in preparation. 55

[198] 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. 55

[199] Xinguo Ren, Florian Merz, Hong Jiang, Yi Yao, Markus Rampp, Hermann Lederer, Volker Blum, and Matthias Scheffler. All-electron periodic G_0W_0 implementation with numerical atomic orbital basis functions: Algorithm and benchmarks. *Phys. Rev. Materials*, 5:013807, Jan 2021. 282

[200] Norina A. Richter, Sabrina Sicolo, Sergey V. Levchenko, Joachim Sauer, and Matthias Scheffler. Concentration of vacancies at metal-oxide surfaces: Case study of mgo(100). *Physical Review Letters*, 111:045502, 2013. 53

[201] S. Ringe, H. Oberhofer, C. Hille, S. Matera, and K. Reuter. Function-space-based solution scheme for the size-modified poisson-boltzmann equation in full-potential dft. *Journal of Chemical Theory and Computation*, 12(8):4052–4066, 2016. 226 , 231

[202] S. Ringe, H. Oberhofer, and K. Reuter. Transferable ionic parameters for first-principles poisson-boltzmann solvation calculations: Neutral solutes in aqueous monovalent salt solutions. *Journal of Chemical Physics*, 146(13):134103, 2017. 226 , 227

[203] H. N. Rojas, R. W. Godby, and R. J. Needs. *Phys. Rev. Lett.*, **74**:1827, 1995. 258

[204] B. Roux. The calculation of the potential of mean force using computer-simulations. *Comput. Phys. Comm.*, 91:275, 1995. 528

[205] R. S. Rowland and R. Taylor. Intermolecular nonbonded contact distances in organic crystal structures: Comparison with distances expected from van der waals radii. *J. Phys. Chem.*, 100:7384–7391, 1996. 374

[206] Victor G. Ruiz, Wei Liu, and Alexandre Tkatchenko. Density-functional theory with screened van der Waals interactions applied to atomic and molecular adsorbates on close-packed and non-close-packed surfaces. *Physical Review B*, 93:035118, 2016. 241 , 245

[207] A. Savin, B. Silvi, and F. Colonna. *Can. J. Chem.*, 74:1088, 1996. 440

[208] A. Sayvetz. *J. Phys. Chem.*, **7**:383, 1939. 167 , 170

[209] C. Schober, K. Reuter, and H. Oberhofer. A critical analysis of fragment orbital dft schemes for the calculation of electronic coupling values. *J. Chem. Phys.*, 144:054103, 2016. 404 , 405 , 503

[210] 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. 64 , 65

[211] K. Senthilkumar, F. C. Grozema, F. M. Bickelhaupt, and L. D. A. Siebbeles. Charge transport in columnar stacked triphenylenes: Effects of conformational fluctuations on charge transfer integrals and site energies. *J. Chem. Phys.*, 119(18):9809–9817, 2003. 404

[212] P. Sherwood, A. H. de Vries de Vries de Vries 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. 204

[213] E. Sigfridsson and U. Ryde. Atomic charges from the electrostatic potential and moments. *J. Comput. Chem.*, 19(4), 1998. 374

[214] U. C. Singh and P. A. Kollman. An approach to computing electrostatic charges for molecules. *J. Comput. Chem.*, 5:129–145, 1984. 374

[215] Markus Sinstein, Christoph Scheurer, Sebastian Matera, Volker Blum, Karsten Reuter, and Harald Oberhofer. *J. Chem. Theory Comput.*, 2017. 210 , 211 , 212 , 213 , 214 , 215 , 220 , 225

[216] Alexey A. Soluyanov and David Vanderbilt. Computing topological invariants without inversion symmetry. *Phys. Rev. B*, 83:235401, Jun 2011. 351

[217] R Strange, FR Manby, and PJ Knowles. Automatic code generation in density functional theory. *Comp. Phys. Comm.*, 136:310–318, 2001. 67

[218] R.E. Stratmann, G.E. Scuseria, and M.J. Frisch. *Chem. Phys. Lett.*, 257:213, 1996. 90

[219] Jianwei Sun, Adrienn Ruzsinszky, and John P. Perdew. Strongly constrained and appropriately normed semilocal density functional. *Phys. Rev. Lett.*, 115:036402, 2015. 65 , 67

[220] C. Tablero. Representations of the occupation number matrix on the LDA/GGA+ U method. *J. Phys. Condens. Matter*, 20(32):325205, 2008. 234

[221] 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. 273

[222] J.D. Talman. NumSBT: a subroutine for calculating spherical bessel transforms numerically. *Comput. Phys. Comm.*, 180(2):332–338, February 2009. 273

[223] J.M. Tao, J.P. Perdew, V.N. Staroverov, and G.E. Scuseria. Climbing the density functional ladder: Nonempirical meta-generalized gradient approximation designed for molecules and solids. *Phys. Rev. Lett.*, 91:146401, 2003. 65

[224] A. Tkatchenko and M. Scheffler. *Phys. Rev. Lett.*, 102:073005, 2009. 62 , 69 , 239 , 240 , 242

[225] Alexandre Tkatchenko, Alberto Ambrosetti, and Robert A DiStasio Jr. Interatomic methods for the dispersion energy derived from the adiabatic connection fluctuation-dissipation theorem. *J. Chem. Phys.*, 138:074106, 2013. 243

[226] Alexandre Tkatchenko, Robert A DiStasio Jr, Roberto Car, and Matthias Scheffler. Accurate and efficient method for many-body van der waals interactions. *Phys. Rev. Lett.*, 108(23):236402, 2012. 243

[227] A Togo and I Tanaka. First principles phonon calculations in materials science. *Scr. Mater.*, 108:1–5, Nov 2015. 497

[228] G. Torrie and J. Valleau. Nonphysical sampling distributions in monte carlo free energy estimation: Umbrella sampling. *J. Comput. Phys.*, 23:187, 1977. 528

[229] L. Triguero, L. G. M. Pettersson, and H. Ågren. Calculations of near-edge x-ray-absorption spectra of gas-phase and chemisorbed molecules by means of density-functional and transition-potential theory. *Phys. Rev. B*, 58:8097–8110, Sep 1998. 201

[230] O. Vahtras, J. Almlöf, and M.W. Feyereisen. *Chem. Phys. Lett.*, 213:514, 1993. 69 , 257

[231] E. van Lenthe van Lenthe van Lenthe van Lenthe, E.J. Baerends, and J.G. Snijders. *J. Chem. Phys.*, 101:9783, 1994. 116

[232] Michiel J. van Setten, Fabio Caruso, Sahar Sharifzadeh, Xinguo Ren, Matthias Scheffler, Fang Liu, Johannes Lischner, Lin Lin, Jack R. Deslippe, Steven G. Louie, Chao Yang, Florian Weigend, Jeffrey B. Neaton, Ferdinand Evers, and Patrick Rinke. Gw100: Benchmarking g0w0 for molecular systems. *Journal of Chemical Theory and Computation*, 11(12):5665–5687, 2015. PMID: 26642984. 259 , 261

[233] S.H. Vosko, L. Wilk, and M. Nusair. *Can. J. Phys.*, 58:1200, 1980. 64 , 65

[234] L. Vočadlo and D. Alfè. *Phys. Rev. B*, 65:214105, 2002. 190 , 191

[235] F. Wagner, Th. Laloyaux, and M. Scheffler. *Phys. Rev. B*, 57:2102, 1998. 133

[236] Homer F. Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4), 2011. 299 , 303

[237] E Weinan, Weiqing Ren, and Eric Vanden-Eijnden. Simplified and improved string method for computing the minimum energy paths in barrier-crossing events. *J. Chem. Phys.*, 126:164103, 2007. 507

[238] J. Wilhelm, M. Walz, M. Stendel, A. Bagrets, and F. Evers. Ab initio simulations of scanning-tunneling-microscope images with embedding techniques and application to c58-dimers on au(111). *Phys. Chem. Chem. Phys.*, 15:6684, 2013. 536

[239] Jianhang Xu, Ruiyi Zhou, Zhen Tao, Christopher Malbon, Volker Blum, Sharon Hammes-Schiffer, and Yosuke Kanai. Nuclear-electronic orbital approach to quantization of protons in periodic electronic structure calculations. *The Journal of Chemical Physics*. 313 , 314 , 320

[240] Yang Yang, Kurt R. Brorsen, Tanner Culpitt, Michael V. Pak, and Sharon Hammes-Schiffer. Development of a practical multicomponent density functional for electron-proton correlation to produce accurate proton densities. *J. Chem. Phys.*, 147:114113, 2017. 315

[241] E. R. Ylvisaker, W. E. Pickett, and K. Koepernik. Anisotropy and magnetism in the LSDA+U method. *Phys. Rev. B*, 79:035103, Jan 2009. 232

[242] Zhi-Qiang You and John M. Herbert. Reparameterization of an accurate, few-parameter implicit solvation model for quantum chemistry: Composite method for implicit representation of solvent, cmirs v. 1.1. *J. Chem. Theory Comput.*, 12(9):4338–4346, 2016. 210

[243] Rui Yu, Xiao Liang Qi, Andrei Bernevig, Zhong Fang, and Xi Dai. Equivalent expression of F_2 topological invariant for band insulators using the non-abelian berry connection. *Phys. Rev. B*, 84:075119, Aug 2011. 351

[244] Igor Ying Zhang, Xinguo Ren, Patrick Rinke, Volker Blum, and Matthias Scheffler. Numeric atom-centered-orbital basis sets with valence-correlation consistency from h to ar. *New Journal of Physics*, 15:123033, 2013. 30 , 35 , 48 , 63 , 70 , 71 , 83 , 96

[245] I.Y. Zhang, N. Su, A.G. Bremond, C. Adamo, and X. Xu. Doubly hybrid density functional xdh-pbe0 from a parameter-free global hybrid model pbe0. *J. Chem. Phys.*, 136:174103, 2012. 63 , 68

[246] Y Zhang, X Xu, and W. A. Goddard. Doubly hybrid density functional for accurate descriptions of nonbond interactions, thermochemistry, and thermochemical kinetics. *Proc. Natl. Acad. Sci. USA*, 106:4963–4968, 2009. 68

[247] Y. Zhang and W. Yang. *Phys. Rev. Lett.*, 80:890, 1998. 64

[248] Y. Zhao and D.G. Truhlar. Density functional for spectroscopy: No long-range self-interaction error, good performance for rydberg and charge-transfer states, and better performance on average than b3lyp for ground states. *J. Phys. Chem. A*, 110:13126, 2006. 67

[249] Y. Zhao and D.G. Truhlar. The m06 suite of density functionals for main group thermochemistry, thermochemical kinetics, noncovalent interactions, excited states, and transition elements: Two new functionals and systematic testing of four m06-class functionals and 12 other functionals. *Theor. Chem. Acc.*, 120:215, 2006. 66 , 67

[250] Y. Zhao and D.G. Truhlar. A new local density functional for main-group thermochemistry, transition metal bonding, thermochemical kinetics, and noncovalent interactions. *J. Chem. Phys.*, 125(19):194101, 2006. 64

[251] Y. Zhao and D.G. Truhlar. Exploring the limit of accuracy of the global hybrid meta density functional for the main-group thermochemistry, kinetics, and noncovalent interactions. *J. Chem. Theor. Comp.*, 4(11):1849, 2008. 67

Following pages: Index

(this page inserted to enforce proper hyperlink to index)

Index