Quick reference manual for the SPHInX DFT program Release 2.7

Christoph Freysoldt, freysoldt@mpie.de

September 8, 2020

Contents

1 Introduction

1.1 About

SPHInX is a C++ programming library for developing efficient physics codes [1]. Its dominant feature is a plane-wave density-functional theory (DFT) code named `sphinx`. This document is a short, non-tutorial manual for the DFT program sphinx. Other features of `sphinx` are not described here. In particular, $\mathbf{k}\cdot\mathbf{p}$, EXX, and tight-binding, are not covered.

1.2 `sphinx` invocation

`sphinx` reads its input from a main input file, usually called '`input.sx`'. Its format is described in Sec. 3. `sphinx` produces output in several dedicated files (see Sec. 6), but the main output comes on stdout (i.e. the screen). `sphinx` can be asked to duplicate (or redirect) its output into a log file. `sphinx` – like all SPHInX executables – can take options on the command line that control some of its global behavior.

Option	argument	description
`--help`		show all the available options
`--log`		create a log file
`--quiet`		don't produce output on stdout
`--input`	filename	input file (default: input.sx)

If SPHInX has been compiled with the FFTW library for the fast Fourier transforms (FFTs), there are two additional options to access the planning mode of FFTW (`--fastfft` and `--wisdom`).

1.3 Parallelism

SPHInX uses MPI (message passing interface) high-level parallelism and openMP low-level parallelism. Both features have to be enabled at compile time!

MPI parallelism allows to distribute memory and workload over independent tasks (processes) which may run even on different computers. At present, SPHInX uses MPI mostly for **k**-point parallelization, parallelization over atoms in the computation of the PAW one-center contributions, and for exact-exchange workload. MPI parallelism is enabled by starting SPHInX with `mpirun`.

Example:

Run `sphinx` with 20 MPI tasks:

```
mpirun -n 20 sphinx
```

openMP parallelism uses multiple threads within the same process, allowing to share the workload among multiple CPU cores on the same computer. openMP is used for low-level parallelization within the numerical libraries (BLAS, FFT) and for some internal routines (e.g. exchange-correlation potential, non-local projectors on x86-64-AVX machines). openMP is enabled by setting the `SX_THREADS` environment variable.

Example:

Run `sphinx` with 4 openMP threads:

```
export SX_THREADS=4
sphinx
```

Note that environment variables in c-shell variants (csh, tcsh) are set differently

```
setenv SX_THREADS 4
sphinx
```

Efficiency. Parallelism always comes at a cost, i.e., the calculation never runs proportionally faster if you provide more cores. This is because there are always some parts that do not run in parallel, AND there is an intrinsic overhead in communication, AND we are wasting some time when waiting for the slowest task to finish. It is impossible to predict how efficiently parallelization works for your calculation – you must test it for a specific system. Yet, some general rules apply

- Efficiency stops when there's nothing to parallelize. If you have 10 **k**-points, no more than 10 MPI tasks can do work.
- We don't use internal load balancing. Efficiency is best if task can be distributed equally (bad example: 15 **k**-points on 7 MPI tasks: a single MPI task must do three **k**-points).
- openMP efficiency levels off well below the total core count. 4-8 openMP threads work reasonably well on modern machines, but not much more.
- Hyperthreading within the CPU is bad for performance. Only use physical cores.
- If you do care about parallelization efficiency, use the detailed timings at the end of the output to determine bottlenecks and how well they parallelize.

MPI and openMP parallelism can be combined when you run into efficiency limits with only one type of parallelization. However, the total core efficiency (i.e. how much of the serial performance of each core you get) is the product of the individual efficiency. So if you have 70% efficiency for MPI and 70% efficiency for openMP, the combination runs at $0.7 \cdot 0.7 = 49\%$ efficiency.

1.4 Units

SPHInX uses mostly atomic units: bohr for coordinates, Hartree atomic units for energies. Some exceptions exist, e.g. for plane-wave cutoffs (Rydberg), electronic temperature (eV), eigenvalues (eV), ...

2 The SPHInX parser

2.1 Overview

The input file format is best explained starting from an **example:**

```
basis  {
   eCut     = 20;   // Ry
   kPoint {
      coords = [1/2,1/2,1/2];
      weight = 1;
      relative;
   }
   folding = 4 * [1,1,1];
}
```

The SPHInX input format is a structured, hierarchical format with a C-like syntax. It consists of named groups and parameters. Parameter and group names are case-sensitive.

Groups. The content of a **group** (such as `basis`) is enclosed in curly brackets {}. Groups may contain parameters, and other groups.

The order of unique groups is *not* important. Groups that may appear multiple times, or groups that describe an action of the code (notably in the `main` group) are processed in the order of appearance.

Parameters. **Parameters** (such as `eCut`) are assigned values with the equal sign. Parameter assignments must be followed by a semicolon (;). Some parameters/variables may be a vector (or a list), for instance `coords` in the example above. The vector elements are comma-separated and enclosed by square brackets [ ]. Parameters may expect string values. The string is enclosed in double quotes ("). Parameters that expect filenames may be enclosed by <> to look for the files in the search path. Enclosure by double quotes looks for them in the current directory.

Flags. Flags are special parameters, that normally do not carry a value. A flag (e.g. `relative`) is set by specifying its name, followed by a semicolon. Flags are unset by assigning the value 0.

White space. White space (including tabs and newlines) can be added quite freely, except within words, numbers, etc.

Comments. Comments can be added by the // and /* */ syntax. A // comment extends until the rest of the line.[1] A /* */ comment omits everything between the /* and */ markers.

2.2 Expressions

The SPHInX parser supports basic algebraic expressions, such as adding, subtracting, multiplying etc. All numbers are changed to double precision when doing so, i.e., `1/2` is equivalent to `0.5`. Some mathematical functions (sqrt, cbrt (cubic root), sin, cos, exp, log) are available. At the top level, additional variables may be set and used in algebraic expressions. Strings can be concatinated with +.

2.3 Included files

The input file may include other files via the include statement.

```
include <parameters.sx>;
include "startStructure.sx";
```

Similar to C/C++, double quotes indicate that the file is expected in the current directory (of execution), while <> indicates to look for the file in the parser's search path. The file `parameters.sx` (located in `share/sphinx/`) contains a large number of predefined keys that may offer mnemonic names for some (numeric) settings. It should always be included.

2.4 The format statement. std files.

Each input file should begin with a `format` statement, e.g. for PAW

```
format paw;
```

What is the role of this statement? The SPHInX input file is read by the SPHInX parser, and validated against a syntax type definition (std). The relevant std file is selected by the format statement in the input file (here, it would load '`paw.std`'. The std files are located[2] in `share/sphinx/std/`. The std files themselves have a similar format as the actual input file, but define the type expected for each parameter, as well as mutual exclusions, range limits, etc. The std files can be an additional ressource for discovering what SPHInX offers. The std files make heavy use of the include feature.

[1]Instead of // one can also use # (like in shell) or % (like in LaTeX).

[2]within the `sphinx/src` folder in the source tree, and within the installation path when installed

3 Overview on the input file for DFT calculations

A typical input file for PAW will look like this:

```
format paw;
include <parameters.sx>;

structure { ... }
basis { ... }
pawPot { ... }
PAWHamiltonian { ... }
initialGuess { ... }
main { ... }
```

The individual groups are described in the other sections of this manual. They define the atomic structure (`structure`), the plane-wave basis set and **k**-points (`basis`), the PAW potentials to be used (`pawPot`), other settings of the PAW Hamiltonian such as the xc functional (`PAWHamiltonian`), how to set up the starting density and wavefunctions (`initialGuess`), as well as the type of calculation (geometry optimization, single-point calculation, band structure, ...) in the `main` group.

A typical input file for norm-conserving pseudopotentials will look like this:

```
format sphinx;
include <parameters.sx>;

structure { ... }
basis { ... }
pseudoPot { ... }
PWHamiltonian { ... }
initialGuess { ... }
main { ... }
```

Quite similar, no? The difference is in the format, the `pseudoPot` and `PWHamiltonian` groups.

3.1 The `structure` group

The structure group specifies the atomic positions in SPHInX format.

Example:

```
structure {
   cell = 10.2 * [[0 ,  1/2,  1/2],
                  [1/2,  0,  1/2],
                  [1/2,  1/2,  0]];
   species {
      element="Si";
      atom { coords = [0 ,0 ,0]; relative; }
      atom { coords = [1/4 ,1/4 ,1/4]; relative; }
   }
}
```

The following parameters may be set:

parameter	description
`movable`	(flag) allow atoms to move. Default: all atoms are movable, unless any movable flag is used for any species/atom.
`movableX`	(flag) allow atoms to move in the x direction. Default: movable, unless movableY or movableZ are used.
`movableY`	(flag) allow atoms to move in the x direction. Default: movable, unless movableX or movableZ are used.
`movableZ`	(flag) allow atoms to move in the x direction. Default: movable, unless movableX or movableY are used.
cell	(required) The unit cell (in bohr). This is a list of the three basis vectors in Cartesian coordinates.

The `movable` flags are applied hierarchically. Settings at the structure level can be overridden at the species or atom level. Disabling a movable flag of a surrounding group is achieved by setting to 0, e.g. `movableY = 0;`

The `structure` group must contain at least one `species` group. It may contain a `symmetry` group.

3.1.1 The `species` group

The `species` group defines atomic positions for one chemical species. Atoms must be sorted by their chemical species.

```
species {
   element="Al";
   atom { ... }
   atom { ... }
}
species {
   element="O";
   atom { ... }
   atom { ... }
   atom { ... }
}
```

The `species` group may contain the `movable`, `movableX`, `movableY`, `movableZ` parameters as specified above. In addition it may contain the `element` parameter to indicate the chemical symbol, enclosed by double quotes.

The `species` group must contain at least one `atom` group.

3.1.2 The `atom` group

The atom group defines atomic positions for one atom. Atoms must be sorted by their chemical species.

The `atom` group may contain the `movable`, `movableX`, `movableY`, `movableZ` parameters as specified above in Sec. 3.1. In addition, the following parameters may be set:

parameter	description
`coords`	(required) The atomic coordinates as a 3-vector. Unless the `relative` flag is employed, the coordinates are Cartesian (in bohr).
`relative`	(flag) The coordinates are given relative to the unit cell vectors.
`movableLine`	(optional) The movement of the atom is restricted to a line. The value gives the direction of the line as a 3-vector.
`label`	(optional string) Assign a label (or rather a tag) to this atom. If labels are used, atoms with different labels are considered inequivalent. Think of spin configurations for a use-case.

3.1.3 `symmetry` group

The `symmetry` group (within the structure group) defines the rotational symmetries of the system around the origin of the coordinate system. If not given, the symmetries are determined automatically. However, non-chemical degrees of freedom (such as spins) may break the symmetry. As all forces / displacements are symmetrized, such a situation may require to set the symmetries by hand. Alternatively, **giving an empty symmetry group switches off symmetrization**.

The `symmetry` group contains multiple `operator` groups. Each `operator` group contains the parameter `S`, a Cartesian rotation matrix given row-wise. The symmetries must form a group.

Example:

```
symmetry {
   // Symmetry Number 1, E
   operator { S = [[1.000000,0.000000,0.000000],
                   [0.000000,1.000000,0.000000],
                   [0.000000,0.000000,1.000000]]; }
   // Symmetry Number 2, m [0,0.707107,-0.707107]
   operator { S = [[1.000000,0.000000,0.000000],
                   [0.000000,0.000000,1.000000],
                   [0.000000,1.000000,0.000000]]; }
}
```

The best way to set up a reduced symmetry group is to use

```
sxstructsym --printsym -i input.sx
```

and then grep the symmetry group from the output and remove the unwanted symmetries.

3.2 The `basis` group

The `basis` group defines the plane-wave basis and the **k**-points, and optionally adjusts FFT mesh sizes.

Example:

```
basis  {
   eCut    = 20;   // Ry
   kPoint {
      coords = [1/2,1/2,1/2];
      weight = 1;
      relative;
   }
   folding = 4 * [1,1,1];
}
```

The `basis` group must contain a `kPoint` (Monkhorst-Pack meshes) or a `kPoints` (bandstructures) group.

The following parameters may be set:

parameter	description
`eCut`	(required) Plane-wave cutoff for wavefunctions (in Rydberg)
`gCut`	(optional) Override cutoff for the density G-basis (defaults to 4× `eCut`)
`folding`	Monkhorst-Pack folding. 3-vector of integers. Defaults to [1,1,1]
`mesh`	(optional) FFT mesh. 3-vector of integers. Default: automatic
`meshAccuracy`	(optional) FFT mesh accuracy, relative to the default. Default: 1, i.e. sufficient to hold 2× the wavefunction plane-waves to avoid wrap-around errors
`saveMemory`	(flag) Do not store phase-factor for each atom. This may make the code marginally slower, but save some RAM.

3.2.1 The `kPoint` group

The `kPoint` group defines the offset of the Monkhorst-Pack mesh. For this, one (or more) k-points are defined within the Brillouin zone. The complete zone with all the **k**-points in it (usually 1) is then shrinked by a factor `folding` in each direction; and the originally Brillouin zone is filled with copies of this shrinked zone. Afterwards, symmetry is used to remove redundant **k**-points.

Alternatively, all k-points can be specified via `kPoint` groups, assigning weights as desired.

Example:

```
kPoint { coords = [1/2,1/2,1/2]; relative; }
```

The following parameters may be set:

parameter	description
`relative`	(flag) Coordinates are relative. This should be usually set.
`coords`	(required) The coordinates. If relative flag is not given, the units are 1/bohr.
`weight`	(optional) Give a weight to this k-point. Weights must sum to 1. Default: 1/number of **k**-points.

Typical offsets are 0 or 1/2. For decoupling across vacuum regions with folding=1, use 1/4 to minimize band dispersion effects.

3.2.2 The `kPoints` group

The `kPoints` group is used to conveniently define band structure paths. It contains a sequence of `from` and `to` groups. **Note**: Band structure paths are subject to Monkhorst-Pack folding, so set folding to [1,1,1] or leave it out.

Example:

```
kPoints {
   dK = 0.01;
   // --- fcc lattice
   // L point
   from { coords=PI/aLat * [1,1,1]; label="L"; }
   // Gamma point
   to { coords=[0,0,0]; label="\xG"; }
   // X point
   to { coords=[2 * PI/aLat, 0,0]; label="X"; }
}
```

The group must contain a `from`, and at least one `to` group.
The following parameters may be set:

parameter	description
`relative`	(flag) Coordinates are relative.
`dK`	(optional) Set the number of intermediate **k**-points such that the distance is at most dK (in 1/bohr).

3.2.3 The `from` group

The `from` group (within the `kPoints` group) adds a single k-point at the desired position. It may be used multiple times.

The following parameters may be set:

parameter	description
`relative`	(flag) Coordinates are relative.
`coords`	(required) The coordinates. If relative flag is not given, the units are 1/bohr.
`label`	(optional) String. Give a label to this k-point.

3.2.4 The `to` group

The `to` group (within the `kPoints` group) adds a line of k-points from the previous one to a new position. The number of points is set directly with `nPoints` or indirectly via `dK`.

The following parameters may be set:

parameter	description
`relative`	(flag) Coordinates are relative.
`coords`	(required) The coordinates. If relative flag is not given, the units are 1/bohr.
`label`	(optional) String. Give a label to this k-point.
`dK`	Set maximum k-point distance.
`nPoints`	Specify number of points to add. The final one will be at `coords`.

3.3 The `pawPot` group

The `pawPot` group defines the PAW potentials, by a sequence of `species` groups. The order of species must agree with the `structure` group.

Example:

```
pawPot {
   species {
      name = "Nitrogen";
      potType = "AbInit";
      element = "N";
      potential="N_LDA_abinit.paw";
      lMaxRho=2;
      angularGrid=4;
   }
   species {
      name = "Hydrogen";
      element="H";
      ...
   }
}
```

The group must contain one `species` group per species. Within the `species` group, the following parameters are commonly set:

parameter	description
`name`	(optional string) English name of the element
`element`	(optional string) Chemical symbol
`potential`	(required filename) Name of the potential file.
`potType`	(required string) Type of the potential file. May be "AbInit", "AtomPAW", "CPPAW", or "VASP".

The following parameters within the `species` group allow experimenting with some details.

parameter	description
`lMaxRho`	(optional) Truncate the spherical expansion of densities (and compensation charges) at this l.
`angularGrid`	(optional integer) Choose a different angular grid for xc calculation in the PAW sphere. Larger is finer. Default is 7 (110 points).
`nRadGrid`	(optional integer) Interpolate to a different radial grid.
`checkOverlap`	(flag) Check that PAW norm is garantueed to be positive definite in the limit of large cutoffs. This is on by default. Some problematic PAW potentials may fail the check, but work normally in some circumstances, so you can switch off the check here.

3.4 The `PAWHamiltonian` group

The `PAWHamiltonian` group defines the DFT functional, the number of empty states, the smearing, and some other settings.

Example:

```
PAWHamiltonian  {
   nEmptyStates = 10;
   ekt = 0.02;
   xc = LDA_PW;
}
```

The following common parameters may be set:

parameter	description
`xc`	(required) xc functional to be used. Constants are defined in `parameters.sx`. See below.
`ekt`	(optional) smearing in eV. Should be 0 for semiconductors.
`MethfesselPaxton`	(optional) If ≥0, use Methfessel-Paxton smearing of indicated order. Order 0 is same as Gaussian smearing.
`FermiDirac`	(optional) If ≥0, use FermiDirac smearing of indicated order. Order 0 is the default; order 1 corresponds to first-order corrections. Higher orders are not yet implemented.
`nEmptyStates`	(optional) number of empty states. Remember to scale with system size! Defaults to zero, which is OK for semiconductors, and catastropic for metals.
`nExcessElectrons`	(optional) Number of extra electrons (= minus charge). Can be fractional.
`spinPolarized`	(flag) Run a collinear spin-polarized calculation.
`dipoleCorrection`	(optional) Use the dipole correction for slab systems. The in-plane lattice must be perpendicular to the z-axis, and the third basis vector must be aligned with the z-axis. For charged calculation, this requests the generalized dipole correction [2], which may need some care for initializing the charge (see `charged` in the `initialGuess` group).
`zField`	(optional) Use an additional electric field along z when using the dipole correction (eV/bohr).

Smearing schemes SPHInX supports Fermi-Dirac (the default), Gaussian, and Methfessel-Paxton smearing. Methfessel-Paxton smearing of order > 0 tends to yield lower electronic entropy at a comparable smearing width compared to Fermi-Dirac and Gaussian smearing and is therefore believed to produce free energies and forces closer to the T=0K values. On the downside, it may produce occupation numbers outside the physical range [0..1] and negative entropies. Note also that high orders introduce wiggles in the occupation function, which may couple to features in the DOS and produce artifacts.

Note that smearing parameters between different smearing schemes are *not* comparable on a 1:1 basis. Taking the full-width at half-maximum (FWHM) value of the underlying distribution as an approximate scaling factor, equivalent smearings require the following relative smearing parameters.

scheme	Fermi-Dirac	FD1	Gaussian(MP0)	MP1	MP2	MP3
FWHM $[k_B T]$	3.53	2.63	1.67	1.25	1.04	0.914
factor	1	1.35	2.11	2.82	3.39	3.9

MPn = Methfessel-Paxton order n, FD1 = Fermi-Dirac order 1.

Example: A Fermi-Dirac smearing of 0.1 eV is approximately equivalent to a Gaussian smearing of 0.21 eV.

Available xc functionals The xc parameter can be LDA_PW (10), i.e. the Perdew-Wang parametrization of LDA, or PBE (1). Perdew-Zunger LDA (0) works, but is not recommended because the parametrization is discontinuous, limiting the convergence in some cases. If you need other functionals, contact `freysoldt@mpie.de`. Hybrid functionals are still experimental, and slow.
The following parameters allow experimenting with some details.

parameter	description
`omegaHSE`	(optional) Change the ω screening length of HSE
`alphaHybrid`	(optional) Change the non-loccal exchange mixing parameter of hybrid functionals.

The `PAWHamiltonian` group may additional contain the `vExt` and `xcMesh` groups to set an external potential and the mesh for xc calculation, respectively. It may also contain a `HubbardU` group for DFT+U calculations. Adding a `vdwCorrection` group enables van-der-Waals corrections.

3.4.1 The `vExt` group

The `vExt` group in the `PAWHamiltonian` or `PWHamiltonian` group defines an external potential.

Example:

```
PAWHamiltonian {
    ...
    vExt { file="sawtooth.sxb"; }
}
```

It contains a single parameter, `file`, which contains the filename of the netcdf-type potential file to be used. The format of that file is like for a density (see Sec. 6.4).

3.4.2 The `xcMesh` group

The `xcMesh` group defines a specific FFT mesh for the calculation of the xc functional. It may be set with either of the following parameters (cf. the mesh definition in the `basis` group). The default is to double the density mesh, roughly equivalent to meshAccuracy=2.

The `xcMesh` group may appear in some groups other than the Hamiltonian to temporally override the xc mesh: `scfDiag.CCG`, `blockCCG`, `CCG`.

parameter	description
`eCut`	Plane-wave cutoff for wavefunctions (in Rydberg). The xc mesh will be sufficient to hold 2× this cutoff.
`mesh`	FFT mesh. 3-vector of integers.
`meshAccuracy`	FFT mesh accuracy, relative to the default. Default: 1, i.e. sufficient to hold 2× the wavefunction plane-waves to avoid wrap-around errors

3.4.3 The `vdwCorrection` group

The `vdwCorrection` group in the `PAWHamiltonian` or `PWHamiltonian` specifies van-der-Waals corrections. At present, only the Grimme D2 implementation has been tested. The Tkatchenko-Scheffler implementation should be considered experimental.

Example:

```
PAWHamiltonian {
   ...
   vdwCorrection { method="D2"; }
}
```

The following parameters may be set:

parameter	description
`method`	(required) The correction method; can be "D2" (Grimme D2) or "TS" (Tkatchenko-Scheffler).
`combinationRule`	(optional) "Tang" or "GB" (Gould-Bucko) for D2.

The default combination rule is due to Tang [3]:

$$C_6^{AB} = \left[\frac{1}{2}\left(\frac{\alpha^B}{C_6^B\alpha^A} + \frac{\alpha^A}{\alpha^B C_6^A}\right)\right]^{-1} ,$$

where C_6 are the C_6 coefficients and α is the polarizability. The alternative is Gould-Bucko [4]

$$C_6^{AB} = 1.43\,\mathrm{Hartree\,bohr}^{-6}(\alpha^A\alpha^B\mathrm{bohr}^6)^{0.725} .$$

3.4.4 The `HubbardU` group

The `HubbardU` group defines on-site correlation corrections from the Hubbard model [5], commonly known as DFT+U. It uses the rotationally invariant formulation for symmetry equivalent orbitals.

Example:

```
PAWHamiltonian {
   ...
   HubbardU {
      site { ... } // 1st element with some U
      site { ... } // 2nd element with some U
      site { ... } // 3rd element with some U
   }
}
```

SPHInX offers three type of sites: deep atomic orbitals using the PAW projector defined via the `site` group, atomic orbitals of a given radial shape via the `AO` group, or molecular orbitals of homonuclear diatomics via the `MO` group. They are described below.

The following parameters may be set:

parameter	description
`verbose`	Request more verbose output (and some test files).

3.4.5 The `site` group

The `site` group within the `HubbardU` group defines on-site correlation corrections using PAW projectors. It is assumed that the PAW projector projects a normalized valence AO to unity.

Example:

```
PAWHamiltonian {
   ...
   HubbardU {
      site { element="Fe"; U = 6; }
   }
}
```

The following parameters may be set:

parameter	description
`element`	defines the element via its name
`species`	defines the element via its species number (1,2,3...) within the input file
`label`	defines the relevant atoms via their label. All atoms must belong to the same species. See also `label` in the `atom` group
`projectorType`	which projector from the PAW potential to use. Starts at 1.
`U`	The actual U value (in eV)
`shift`	An additional energy shift of the projector (in eV)

The species must be uniquely defined via `element`, `species`, or `label`.

3.4.6 The `AO` group

The `AO` group within the `HubbardU` group defines on-site correlation corrections using AO orbital projectors. The projectors are constructed from atomic orbitals (AOs) of given radial shape. This shape is defined in the `orbital` group. The AO projectors include also the atomic PAW normalization.

Example:

```
PAWHamiltonian {
   ...
   HubbardU {
     AO {
         element="O";
         orbital { file="quamol-O.sxb"; is=1; iot=1; }
         rCut = 3;
         cutWidth=0.5;
         U = 0.5;

         nRadGrid=100;
     }
   }
}
```

The group must contain an `orbital` group.

The following parameters may be set:

parameter	description
`element`	defines the element via its name
`species`	defines the element via its species number (1,2,3...) within the input file
`label`	defines the relevant atoms via their label. All atoms must belong to the same species. See also `label` in the `atom` group
`nRadGrid`	number of radial points to represent atomic orbital projector. Default: 200
`rCut`	cutoff radius for atomic orbital (in bohr)
`cutWidth`	(optional) smoothening parameter for cutoff (in bohr). Default 0.7 bohr.
`U`	The actual U value (in eV)
`shift`	An additional energy shift of the projector (in eV)

The species must be uniquely defined via `element`, `species`, or `label`.

3.4.7 The `MO` group

The `MO` group within the `HubbardU` group defines on-site correlation corrections using MO orbital projectors. The molecular orbitals (MOs) are constructed from atomic orbitals (AOs) of given radial shape. This shape is defined in the `orbital` group. The MO projectors are constructed from AO projectors such that a normalized MO is projected to unity. The AO projectors include also the atomic PAW normalization.

Example:

```
PAWHamiltonian {
   ...
   HubbardU {
     MO {
         element="O";
         orbital { file="quamol-O.sxb"; is=1; iot=1; }
         mMO = 1;
         sign=-1;
         rCut = 3;
         cutWidth=0.5;
         U = 0.5;
         minDist = 2.0;
         maxDist = 3.5;

         nRadGrid=100;
         nInterpolate=10;
     }
   }
}
```

The group must contain an `orbital` group.
The following parameters may be set:

parameter	description
`element`	defines the element via its name
`species`	defines the element via its species number (1,2,3...) within the input file
`label`	defines the relevant atoms via their label. All atoms must belong to the same species. See also `label` in the `atom` group
`maxDist`	(required) maximum distance (in bohr) of two atoms to be considered a molecule
`minDist`	(optional) minimum distance (in bohr) of two atoms to be considered a molecule. Defaults to 50% of maxDist.
`nInterpolate`	(optional) number of distance points used to interpolate orbital normalization. Default: 100
`nRadGrid`	number of radial points to represent atomic orbital projector. Default: 200
`rCut`	cutoff radius for atomic orbital (in bohr)
`cutWidth`	(optional) smoothening parameter for cutoff (in bohr). Default 0.7 bohr.
`mMO`	rotational constant of orbital symmetry (σ=0, π=1)
`sign`	relative sign of orbitals on both atoms. Can be +1 or -1.
`U`	The actual U value (in eV)
`shift`	An additional energy shift of the projector (in eV)

The species must be uniquely defined via `element`, `species`, or `label`.

3.4.8 The `orbital` group

The `orbital` group within the `HubbardU` { `MO` {}} or `HubbardU` { `AO` {}} group defines where the radial shape of the atomic orbital is found. This is usual a quamol [6], an optimized orbital for the molecular species at hand.

Example:

```
PAWHamiltonian {
   ...
   HubbardU {
      MO {
         ...
         orbital { file="quamol-O.sxb"; is=1; iot=1; }
      }
   }
}
```

The following parameters may be set:

parameter	description
`fromPotential`	(flag) Get orbital shape from potential. This is not recommended.
`file`	(filename) Get orbital shape from this quamol-type sxb file.
`is`	(optional) species id within file (starts at 0). If not given, assumes same species ordering in sxb file as in input file.
`iot`	(required) Which orbital to take. Starts at 0.

3.5 The `spinConstraint` group

The optional `spinConstraint` group defines atomic spin constraints for PAW calculations. They can be set via atomic labels, or read in for all atoms from an external file.

Note: Atomic spins refer to the spin density inside the PAW cutoff sphere and are not identical to the definition used for setting up spin configurations from complete atoms in `initialGuess.atomicSpin`.

Example:

```
spinConstraint  {
   label ="No.1";
   constraint = 2;
}
spinConstraint  {
   label ="No.2";
   constraint = -2.5;
}
```

The following parameters may be set:

parameter	description
`label`	The present constraint applies to atoms with the given label.
`constraint`	Value of the desired atomic spin
`file`	Read all spin constraints from this file.

If the constraints are read from a file, there must be one number per atom. If not all atoms are constrained, the number is to be replaced by 'X', followed by a newline. For instance,

```
1
1
X
-2
```

for a four-atom calculation would constrain the spins of the first two atoms to a value of 1, the last one (no. 4) to a value of -2, and have no spin constraint on atom 3.

3.6 The initialGuess group

In order to start a DFT calculations, one must set up an initial guess for the density and for the wave functions. The `initialGuess` group defines how this is done, as well as a few other settings (such as keeping the waves on disk to save RAM).

Example:

```
initialGuess   {
   waves { lcao {} }
   rho { atomicOrbitals; }
}
```

The default is to set up the density from a superposition of atomic densities, and the wave-functions from a single-step LCAO calculation, using the atomic valence orbitals [7]. This works exceptionally well. If you want to finetune the behavior, the `initialGuess` group must contain a `waves` or a `rho` group. Otherwise, you may omit the `waves` and `rho` groups to get the default behavior.

Additionally, the `initialGuess` group may contain an occupations group to set up initial occupations (notably when keeping them fixed), and an `exchange` group for hybrid functionals.

The following parameters may be set:

parameter	description
`noWavesStorage`	(flag) Do not save wavefunctions after the initial guess
`noRhoStorage`	(flag) Do not save the density after the initial guess.

3.6.1 The waves group

The `waves` group defines the strategy for setting up the initial waves. The major strategies are LCAO (best), reading from a file (excellent, if you have one), or random. To use LCAO, you must include an `lcao` group.

The following parameters may be set:

parameter	description
`file`	(optional) File name for reading in the initial guess from a previous calculation.
`random`	(flag) Initialize with random numbers.
`keepWavesOnDisk`	(flag) Keep waves on disk, load only a single **k**-point at each time. May save a lot of RAM, but can be quite a bottleneck on small systems.

3.6.2 The lcao group

The `lcao` group within the `initialGuess.waves` group finetunes the LCAO calculation, if necessary. Notably, you can iterate the LCAO calculation to self-consistency. This is generally **no good idea**.

parameter	description
`maxSteps`	(optional) Max. number of steps. If 0 or 1, the initial density will *not* be updated.
`dEnergy`	(optional) Energy convergence criterium

In addition to these, the `lcao` group supports all the mixer settings (`mixingMethod, nPulaySteps, spinMixing, rhoMixing`, `preconditioner` group) of the `scfDiag` group (Sec. 4.1).

3.6.3 The `rho` group

The `rho` group defines the strategy for the initial density. This can be a superposition of atomic densities, from a file, from the wave functions (if read from a file), or random.

The following parameters may be set:

parameter	description
`file`	Read density from this file.
`fromWaves`	(flag) Compute from the wave functions (which must be from file in this case).
`random`	(flag) Request random density
`atomicOrbitals`	(flag) Superposition of atomic densities
`spinMoment`	(optional) When from atomic densities, apply a global spin polarization.

If `atomicOrbitals` is chosen, one may set up a spin configuration by using `atomicSpin` groups.

For charged calculations, one may specify an initial localization of charge in the `charged` group.

3.6.4 The `atomicSpin` group

The `atomicSpin` group (within the `initialGuess.rho` group) defines spin polarization for the initial guess on an per-atom basis when the initial density is set up from atoms. Post-polarizing an external density is not possible at present. Atoms can be identified per label (one `atomicSpin` group per label), or from an external file with spins for all atoms.

Example:

```
aLat = 5.35;
structure {
   include <structures/sc.sx>;
   species { element="Fe";
      atom { coords=[0,0,0]; relative; label="A"; }
      atom { coords=[1/2, 1/2, 1/2]; relative;
             label="B"; }
   }
}
initialGuess {
   rho {
      atomicOrbitals;
      atomicSpin { label="A"; spin=2; }
      atomicSpin { label="B"; spin=-2; }
   }
}
```

The following parameters may be set:

parameter	description
`spin`	The desired spin moment value.
`label`	For which atoms does this spin apply?
`file`	(excludes spin and label) Read atomic spins from this file (one spin per line), one per atom, in sequential order.

3.6.5 The `charged` group

The `charged` group (within the `initialGuess.rho` group) defines the localization of initial charge. This may be a Gaussian charge, or a sheet-layer charge along z (for charged slabs) with a Gaussian profile.

Example:

```
initialGuess {
   rho {
      atomicOrbitals;
      charged { charge=1; coords=[0,0,13.2]; beta=2; }
   }
}
```

The following parameters may be set:

parameter	description
`charge`	(required) The classical charge (i.e. -nExcessElectrons from the `PAWHamiltonian` or `PWHamiltonian` group).
`beta`	(optional) Gaussian broadening
`z`	Request a sheet charge at this z
`coords`	Request a Gaussian charge at this position (in bohr).

Multiple charges can be set up. There is no check if the resulting density has the correct number of electrons.

3.6.6 The `occupations` group

The `occupations` group within the `initialGuess` group defines the initial occupations. This makes sense if the density is computed from wave functions, or if the occupations are going to be fixed at these values.

Example:

```
initialGuess {
   ...
   occupations {
      bands { range=[1,10]; focc=2; }
      kPoints { values=[1,3];
         bands { values=[10]; focc=0; }
         bands { values=[11]; focc=2; }
      }
   }
}
```

One can specify selected sets of states by nesting the following groups: `kPoints`, `spin`, `bands`. Each of these groups selects one or more indices (starting at 1) from the specified index type (i.e. **k**-points, spins, states) by one of the following two parameters:

parameter	description
`values`	(list of integers) Specifically list the indices affected.
`range`	(list of 2 integers) Specify start and end index.

The possible nestings are

- `kPoints { spin { bands { ... focc=...; } } }`
- `kPoints { bands { ... focc=...; } }`
- `spin { bands { ... focc= ...; } }`
- `bands { ... focc=...; }`

The innermost loop must set the `focc` parameter to the desired occupation. If an outer group is omitted, the settings apply to all indices of that type (e.g. no `kPoints` group = all **k**-points). The groups are processed in order of appearance, i.e., later groups override the previous settings, if the index ranges overlap.

In the above example, the first `bands` group sets the occupations of bands 1–10 at all **k**-points to 2. Then, we change the occupation at **k**-points 1 and 3, to have zero occupation in band 10 and an occupation of 2 for band 11.

3.6.7 The `exchange` group

Note: hybrid functionals are experimental and slow.

The `exchange` group allows to set waves for the non-local exchange operator at the initialization stage. This is necessary if you want to initialize the waves from an LCAO calculation. The exchange group contains a single parameter, `file`, which contains the filename of the waves file to be used.

Example:

```
initialGuess {
   exchange { file="waves-pbe.sxb"; }
   rho { file="rho-pbe.sxb"; }
   waves { lcao {} }
}
```

3.7 The `pseudoPot` group

The `pseudoPot` group defines the norm-conserving pseudopotentials by a sequence of species groups. The order of species must agree with the structure group.

Note: PAW and norm-conserving pseudopotentials cannot be mixed. Using `pseudoPot` requires to use `PWHamiltonian` to define the Hamiltonian.

Example:

```
pseudoPot {
   species {
      potential        = "al.cpi";
      name             = "Aluminum";
      element          = "Al";
      valenceCharge    = 3;
      lMax             = d;
      lLoc             = d;
      lcaoOrbitals     = [s,p,d];
      atomicRhoOcc     = [2,1,0];
      rGauss           = 1;
      reciprocalMass   = 26.98;
      dampingMass      = 0.7;
      ionicMass        = 26.98;
   }
}
```

The group must contain one `species` group per species. Within the `species` group, the following parameters may be set:

parameter	description
`name`	(required string) English name of the element
`element`	(optional string) Chemical symbol
`potential`	(required filename) Name of the potential file.
`valenceCharge`	(required) Valence charge.
`lMax`	(required) Max. l-component of the pseudopotential.
`lLoc`	(required) Select the local l-component.
`lcaoOrbitals`	(required) Which orbitals should be used for lcao initialization. **Note:** `s,p,d`, and `f` are predefined constants in `parameters.sx`.
`rGauss`	(required) Broadening of compensation charge, usually 1.
`atomicRhoOcc`	(required) Occupation numbers for charge initialization.
`reciprocalMass`	(required) Mass of the ion.
`dampingMass`	(required,currently unused). Damping for damped-Newton geometry optimization.
`ionicMass`	(required) Mass of the ion.

Note: The appearance of two masses, as well as the lack of default parameters is a historic feature (mistake?) and may be changed in future versions.

3.8 The `PWHamiltonian` group

The `PWHamiltonian` group defines the DFT functional, the number of empty states, the smearing, and some other settings.

Example:

```
PWHamiltonian  {
   nEmptyStates = 10;
   ekt = 0.02;
   xc = LDA;
}
```

The following common parameters may be set:

parameter	description
`xc`	(required) xc functional to be used. Constants are defined in `parameters.sx`. See below.
`ekt`	(optional) smearing in eV. Should be 0 for semiconductors.
`MethfesselPaxton`	(optional) If ≥ 0, use Methfessel-Paxton smearing of indicated order. Order 0 is same as Gaussian smearing.
`FermiDirac`	(optional) If ≥ 0, use FermiDirac smearing of indicated order. Order 0 is the default; order 1 corresponds to first-order corrections. Higher orders are not yet implemented.
`nEmptyStates`	(optional) number of empty states. Remember to scale with system size! Defaults to zero, which is OK for semiconductors, and catastropic for metals.
`nExcessElectrons`	(optional) Number of extra electrons (= minus charge). Can be fractional.
`spinPolarized`	(flag) Run a collinear spin-polarized calculation.
`dipoleCorrection`	(optional) Use the dipole correction for slab systems. The in-plane lattice must be perpendicular to the z-axis, and the third basis vector must be aligned with the z-axis. For charged calculation, this requests the generalized dipole correction, which may need some care for initializing the charge (see `charged` in the `initialGuess` group).
`zField`	(optional) Use an additional electric field along z when using the dipole correction (eV/bohr).

For smearing, please see notes on smearing schemes in the `PAWHamiltonian` section on page 14.

Available xc functionals The xc parameter can be LDA_PW (10), i.e. the Perdew-Wang parametrization of LDA, or PBE (1). Perdew-Zunger LDA (0) works, but is not recommended because the parametrization is discontinuous, limiting the convergence in some cases. If you need other functionals, contact `freysoldt@mpie.de`.

The `PWHamiltonian` group may additional contain the `vExt` and `xcMesh` groups to set an external potential and the mesh for xc calculation, respectively. Adding a `vdwCorrection` group enables van-der-Waals corrections.

3.9 The `main` group

The `main` group selects and controls the algorithms after startup during the main part of the calculation. The can be divided into two categories: electronic minimization (Sec. 4) and geometry optimization (Sec. 5). The main group can contain a single algorithm, or a sequence of them, which are then executed in order of appearance.

All self-consistent DFT calculations must employ some algorithm to compute the electronic wavefunctions and density for a given geometric structure. If additionally the atomic positions are altered according to the calculated forces, electronic minimization becomes the inner loop. Consequently, the electronic minimizers appear in the `bornOppenheimer` group of the geometry optimizers. Electronic minimizers may appear as a direct subgroup of `main`. It may sometimes be useful to select a different set of electronic minimization settings for the initial phase, if the initial guess is not good enough, and then use other settings (less robust, more efficient) later on.

Example:

```
main {
    // --- get a better starting guess with
    //      small rhoMixing
    scfDiag {
        dEnergy=1e-4;
        rhoMixing=0.3;
        maxSteps=100;
    }
    // --- optimize geometry
    linQN {
        dEnergy = 1e-6;
        maxSteps=50;
        bornOppenheimer {
            scfDiag {
                dEnergy = 1e-8;
                rhoMixing=1;
            }
        }
    }
}
```

The group must contain one or more electronic minimizer (`scfDiag`, `CCG`) or geometry optimizer (`QN`, `linQN`, `ricQN`) group.

The main group has no parameters of its own.

4 Electronic loop

4.1 `scfDiag`: iterative diagonalization + density mixing

The `scfDiag` group selects and controls the iterative diagonalization + density mixing algorithm for the solution of the Kohn-Sham DFT equations.

Example:

```
scfDiag {
   blockCCG { blockSize=32; maxStepsCCG=4; }
   dEnergy   = 1e-7;
   preconditioner { type=KERKER; scaling=1;
                    kerkerDamping=1; }
   mixingMethod   = PULAY;
   nPulaySteps    = 7;
}
```

The group must contain one of the iterative diagonalization groups: `CCG` (conjugate-gradient[3]), `blockCCG` (block conjugate-gradient, recommended). It may contain a `xcMesh` group (to override the xc mesh) and a `preconditioner` group (for the density). If the density preconditioner is not specified, Kerker preconditioning with a default damping is used.

The following parameters may be set:

[3]The first C stands for *Complex*.

parameter	description
`dEnergy`	(optional) Free energy convergence criterium (in Hartree). Default is 10^{-8}.
`maxSteps`	(optional) Max. number of steps (density updates).
`maxResidue`	(optional) Additional requirement for convergence: density residue must fall below this threshold.
`printSteps`	(optional) Eigenvalues are printed by default every 10 steps. This interval can be changed by `printSteps`.
`mixingMethod`	(optional) Method for the density mixing. Constants defined in `parameters.sx`. Can be PULAY (default, 2) or LINEAR (0).
`nPulaySteps`	(optional) Number of previous steps (densities) to use in Pulay mixing. Default: 7.
`rhoMixing`	(optional) Additional linear mixing factor for density updates (1=full update (default), 0=no change). Low values may lead to a more stable convergence, but will slow down the calculation if set too low.
`spinMixing`	(optional) Linear mixing parameter for spin densities.
`keepRhoFixed`	(flag) Do not update the density (for band structures).
`keepOccFixed`	(flag) Do not update the occupation numbers.
`keepSpinFixed`	(flag) Do not change the spin moment.
`spinMoment`	(optional) Keep the spin moment at this value.
`ekt`	(optional) Override electronic temperature setting in the Hamiltonian group.
`dipoleCorrection`	(flag) Override the dipole correction setting in the Hamiltonian group.
`dSpinMoment`	accuracy of iterative enforcement of spin constraints. Default: 10^{-8}.
`noRhoStorage`	(flag) Do not write `rho.sxb`.
`noWavesStorage`	(flag) Do not write `waves.sxb`

Multiple iterative diagonalizer groups may be used to change the xc mesh during the minimization. For this, each diagonalizer groups must contain the `dEnergy` parameter. **Example:**

```
scfDiag {
   blockCCG { xcMesh { meshAccuracy=0.7; }
              dEnergy=1e-4; }
   blockCCG { xcMesh { meshAccuracy=1; }
              dEnergy=1e-8; }
   blockCCG { xcMesh { meshAccuracy=2; }
              dEnergy=1e-9; }
   dEnergy=1e-9;
}
```

This example would use an xc-mesh with a relative accuracy of 0.7 until the energy has converged to 10^{-4} Hartree, a standard 2× mesh (accuracy=1) until the energy change is below 10^{-8} Hartree, and a double-accuracy mesh thereafter. The following parameters allow messing around with some details.

parameter	description
`dRelRes`	(optional) The energy convergence criterium for eigenvalues is calculated from the density residue adaptively. This parameter allows to set the initial (and maximum) value of the scaling factor.
`calcForces`	(flag) Calculate and print forces in each step (slow!). This can be used to determine the influence of convergence criteria on the precision in forces.
`dumpTime`	(optional) Set interval between dumps of density/wave functions. The dumps may allow for restarts when a calculation crashes. Since the dumping happens at the end of a SCF cycle, the actual dumping intervals may be longer. Default: 12 hours.

4.1.1 The `CCG` group

The `CCG` group (within the `scfDiag` group) selects conjugate-gradient algorithm for (inner-loop) iterative diagonalization. After all states have been updated, a subspace diagonalization is performed. This algorithm works best for very small systems. For larger systems (> 5 states), the `blockCCG` is superior.

Note: `CCG` is also the name for the conjugate-gradient-based direct minimization algorithm, see Sec. 4.2.

The following parameters may be set:

parameter	description
`dRelEps`	(optional) Stop iterating when the change in eigenvalue falls below this fraction of the change in the first (steepest-descent) step.
`maxStepsCCG`	(optional) Max. number of steps to perform. Default: 5.
`dEnergy`	(optional) Use these settings until energy change fall below this threshold.

The `CCG` group may contain a `xcMesh` group to override the xc mesh settings from the Hamiltonian or the `scfDiag` group.

4.1.2 The `blockCCG` group

The `blockCCG` group (within the `scfDiag` group) selects the block conjugate-gradient algorithm for (inner-loop) iterative diagonalization. After all states have been updated, a subspace diagonalization is performed. For very small systems (≤5 states), the unblocked `CCG` is superior.

The following parameters may be set:

parameter	description
`dRelEps`	(optional) Stop iterating when the change in eigenvalue falls below this fraction of the change in the first (steepest-descent) step.
`maxStepsCCG`	(optional) Max. number of steps to perform. Default: 5.
`blockSize`	(optional) Block size. Default: 64
`nSloppy`	(optional) Don't try to converge the highest nSloppy states (useful for bandstructures).
`dEnergy`	(optional) Use these settings until energy change fall below this threshold.
`verbose`	(flag) Produce debug output.
`numericalLimit`	(flag) Stop iterating when approaching the numerical limit.

The `blockCCG` group may contain a `xcMesh` group to override the keyxc mesh settings from the Hamiltonian or the `scfDiag` group.

4.1.3 The `preconditioner` group

The `preconditioner` group defines the density preconditioner, i.e., a transformation of the observed (or predicted) difference between the input and output density to the applied changes to the input density. An ideal preconditioner models the screening behavior of the system and is able to include the expected screening response into the suggested density change. Selecting an appropriate preconditioner, that reflects the screening properties of the system at hand, is a key to an efficient (i.e. fast) convergence. The preconditioner does *not* affect the converged result.

The following parameters may be set:

parameter	description
`type`	(required) Type of preconditioner, see below
`scaling`	(optional) Additional scaling factor. Default 1.
`spinScaling`	(optional) Additional scaling factor for the spin density. Default 1.
`kerkerDamping`	(optional) Damping constant for Kerker preconditioner. Default: 0.6
`dielecConstant`	(optional) Dielectric constant for the CSRB model.

The following preconditioner types are available (constants defined in `parameters.sx`).

- NONE (0). No preconditioning. Ideal for atoms/molecules in vacuum.
- KERKER (1). Kerker preconditioner. Ideal for metals.
- CSRB (3). Preconditioner for semiconductors based on the Cappellini-del-Sole-Reining-Bechstedt model dielectric function. Requires `dielecConstant`.
- ELLIPTIC (5). An explicit-solver preconditioner. No screening in vacuum region, Thomas-Fermi screening (Kerker-like) elsewhere. Ideal for metallic

slabs.

4.2 CCG: direct minimization

The `CCG` group selects and controls the direct minimization algorithm for the solution of the Kohn-Sham DFT equations [8].

Example:

```
CCG {
   dEnergy=1e-7;
   finalDiag;
}
```

It may contain a `xcMesh` group to override the xc mesh settings from the Hamiltonian group.

The following parameters may be set:

parameter	description
`dEnergy`	(optional) Free energy convergence criterium (in Hartree). Default is 10^{-8}.
`maxSteps`	(optional) Max. number of steps.
`printSteps`	(optional) Eigenvalues are printed by default every 10 steps. This interval can be changed by `printSteps`.
`initialDiag`	(flag) Perform iterative wave-function optimization based on the initial density (this is the default)
`finalDiag`	(flag) Perform subspace diagonalization at the end.
`kappa`	(optional) Initial mixing between subspace Hamiltonian and wave-function updates. If set to a negative value, the value of κ will be fixed at the absolute value. Otherwise, κ is adapted on the fly.
`keepOccFixed`	(flag) Do not update the occupation numbers.
`ekt`	(optional) Override electronic temperature setting in the Hamiltonian group.
`dipoleCorrection`	(flag) Override the dipole correction setting in the Hamiltonian group.
`noRhoStorage`	(flag) Do not write `rho.sxb`.
`noWavesStorage`	(flag) Do not write `waves.sxb`

Note about switching between `scfDiag` and `CCG`: The direct minimization algorithm requires a good initial guess for the wave functions. Therefore, the default is to run a iterative diagonalization (blockCGG algorithm) before starting direct minimization. This is unnecessary when the wavefunctions come from iterative diagonalization, and can be switched off via the `initialDiag` flag. For semiconductors with no empty states, on the other hand, the wavefunctions spanning the occupied subspace may not be diagonalizing the Hamiltonian. In order to switch to `scfDiag` or to obtain eigenvalues, the `finalDiag` flag must be set. If partially occupied or empty states are computed, an approximate diagonalization takes place as part of the algorithm.

4.3 `evalForces`: force evaluation

The `evalForces` group **does not refine the wave-functions**, but is used to calculate forces and write them to a file in sx-format. This is useful for single-point calculations without a structure optimization. It should be used after an electronic loop.

Example:

```
CCG {
   dEnergy=1e-7;
}
evalForces { file="forces.sx"; }
```

The following parameters may be set:

parameter	description
`file`	(optional) Filename.

If the group is encountered multiple times (or multiple groups write to the same file), the additional force output will appended. The output format is in `relaxHist.sx` format.

5 Geometry optimizers

This section describes the geometry optimization groups. We have experimented with geometry optimization and found BFGS quasi-Newton schemes to be most efficient. Therefore, you will see three quasi-Newton optimizers. For small systems, the performance is quite similar. For more complex systems, the `ricQN` variant is recommended.

In addition to the geometry optimizers, there is a pseudo-optimizer `extControl`. This algorithm opens two communication channels (named pipes) and allows to control geometry updates (and other things) from external scripts.

In all cases, the inner electronic loop enjoys the same flexibility of sequencing as the `main` group. Therefore, each geometry optimizer group contains a `bornOppenheimer` group that contains one or more electronic minimizers.

5.1 The `QN` group

The `QN` group selects and controls the geometry optimization via quasi-Newton scheme with BFGS updates. **Note:** In general, `ricQN` is the faster algorithm.

Example:

```
QN {
    maxSteps = 20;
    dEnergy = 1e-5;
    dF = 1e-3;
    maxStepLength=0.2;
    bornOppenheimer {
        scfDiag {
            maxSteps = 50;
            blockCCG { }
            dEnergy = 1e-7;
        }
    }
}
```

The group must contain a `bornOppenheimer` group to specify the electronic loop.

The following parameters may be set:

parameter	description
`maxSteps`	(optional) max. number of steps, default: 50
`dX`	(optional) convergence reached only when maximum displacement (length of displacement vector of a single atom) is less than this value (in bohr). Default: 0.01
`dF`	(optional) convergence reached only when maximum force (length of force vector of a single atom) is less than this value (in Hartree/bohr). Default: 0.001
`dEnergy`	(optional) convergence reached only when change in energy is less than this value (in Hartree). Default: 10^{-4}
`maxStepLength`	maximum allowed displacement (length of displacement vector for a single atom) in bohr. Larger steps are reduced by scaling. Default: 0.3
`hessian`	(filename) Initialize Hessian from file.
`driftFilter`	(flag) Project out the average force and displacement. Default: yes, if no constraints are used.

5.1.1 The `bornOppenheimer` group

The `bornOppenheimer` group defines the electronic loop within a geometry optimization. It contains one or more of the electronic loop groups, see Sec. 4. If more than one minimizer is used, the complete electronic loop sequence is executed at each ionic step.

5.2 The `linQN` group

The `linQN` group selects and controls the geometry optimization via quasi-Newton scheme with BFGS updates for the inverse Hessian. **Note:** In general, `ricQN` is the faster algorithm.

Example:

```
linQN {
    maxSteps = 20;
    dEnergy = 1e-5;
    dF = 1e-3;
    maxStepLength=0.2;
    bornOppenheimer {
       scfDiag {
          maxSteps = 50;
          blockCCG { }
          dEnergy = 1e-7;
       }
    }
 }
```

The group must contain a `bornOppenheimer` group to specify the electronic loop.

The following parameters may be set:

parameter	description
`maxSteps`	(optional) max. number of steps, default: 50
`dX`	(optional) convergence reached only when maximum displacement (length of displacement vector of a single atom) is less than this value (in bohr). Default: 0.01
`dF`	(optional) convergence reached only when maximum force (length of force vector of a single atom) is less than this value (in Hartree/bohr). Default: 0.001
`dEnergy`	(optional) convergence reached only when change in energy is less than this value (in Hartree). Default: 10^{-4}
`maxStepLength`	maximum allowed displacement (length of displacement vector for a single atom) in bohr. Larger steps are reduced by scaling. Default: 0.3
`nProjectors`	(optional) number of previous steps to use for BFGS updates. Default: 10
`hessian`	(filename) Initialize Hessian from file.
`driftFilter`	(flag) Project out the average force and displacement. Default: yes, if no constraints are used.

5.3 The `ricQN` group

The `ricQN` group requests a quasi-Newton optimization with BFGS updates [9] of an on-the-fly optimized internal-coordinate based initial guess for the Hessian.

The following parameters may be set:

parameter	description
`maxSteps`	(optional) max. number of steps, default: 50
`dX`	(optional) convergence reached only when maximum displacement (length of displacement vector of a single atom) is less than this value (in bohr). Default: 0.01
`dF`	(optional) convergence reached only when maximum force (length of force vector of a single atom) is less than this value (in Hartree/bohr). Default: 0.001
`dEnergy`	(optional) convergence reached only when change in energy is less than this value (in Hartree). Default: 10^{-4}
`nProjectors`	(optional) number of previous steps to use for BFGS updates. Default: 10
`maxStepLength`	maximum allowed displacement (length of displacement vector for a single atom) in bohr. Larger steps are reduced by the trust radius method to a value close to the maximum (within 1%). Default: 0.3
`softModeDamping`	(optional) Initial value for Hessian shift (in Hartree/bohr^2). This is overriden with the first successful fit of a positive shift parameter. Default: 1e-2.
`driftFilter`	(flag) Project out the average force and displacement. Default: yes, if no constraints are used.

The group must contain a `bornOppenheimer` group to specify the electronic loop. The `ricQN` group may contain a `ric` group (see Sec. 5.3.1 to define the internal coordinate generation. If left out, default parameters are used, and output from the internal coordinate setup is suppressed.

5.3.1 The `ric` group

The `ric` group defines the parameters for internal coordinate generation.

The following parameters may be set:

parameter	description
`maxDist`	maximum possible distance for considering neighbors (in bohr). Default: 10
`typifyThreshold`	minimum bond length separation of distinct bond types (the f parameter in [9]). After sorting the bond lengthes, the logarithm of subsequent lengthes are compared. If they differ by less than the threshold, the two bonds are assigned the same bond type. Default: 0.05
`rmsThreshold`	minimum distance between two bond length clusters in units of their root-mean-square displacements (the R parameter of [9]). Default: 3
`planeCutLimit`	Relative size of coordination polyhedra to separate the nearest neighbors from further atoms (the P parameter of [9]). Larger values allow for more neighbors. Default: 0.95
`withAngles`	(flag) add bond angle coordinates for all bonds
`bvkAtoms`	(optional, experimental) List of atom ids (starting from 1) for which born-von-Karman transversal force constants are added. The comma-separated list must be enclosed by square brackets [ ]. This adds a bond-directional coordinate to each bond of the atoms in the list.

5.4 The `extControl` group

The `extControl` group allows to control parts of the SPHInX run by external scripts. For this, two communication channels are opened. Typically, these will be named pipes. For more info on the concept, see the `sxextopt` manual. The `extControl` group is the 'DFT code' side of the atomic structure algorithm protocol (ASAP).

The names of the communication files are specified via the environment, in `SX_EXT_CTRL` and `SX_EXT_RES`, respectively.

The group must contain one or more `bornOppenheimer` groups. Each `bornOppenheimer` group may contain an `id` parameter than allows to run the specified sequence of electronic minimizers (usually one) with the `run` command (see below).

If forces are not needed, the `noForces` flag can be used to suppress the calculation of forces.

The extControl algorithm understands the following commands:

command	description
`confirm y`	Confirm *ASAP* commands
`confirm n`	Do not confirm *ASAP* commands
`end`	End the calculation
`run [<id>]`	Run specified bornOppenheimer group (first one, if `<id>` is omitted).
`onproblem=crash`	Crash on wrong commands
`onproblem=stop`	Stop on wrong commands
`onproblem=ignore`	Ignore wrong commands (default)
`get energy`	Print free energy on result channel
`get forces`	Print forces on result channel
`get natoms`	Print number of atoms on result channel
`get nspecies`	Print number of species, and for each species number of atoms on result channel
`get structure`	Print atomic coordinates on result channel
`get positions`	Print atomic coordinates + chemical symbol on result channel
`get cell`	Print unit cell on result channel
`set structure`	Set atomic coordinates
`set positions`	Set atomic coordinates
`shift atom`	`<id> <dx> <dy> dz>` shift specified atom. id starts at 1 and enumerates atoms across all species.
`set cell`	*not supported*
`get stress`	*not supported*
`get elements`	Print chemical symbols on result channel
`get nspinconstraints`	Print number of spin constraints on result channel
`get nu`	Print spin constraint Langrange parameters on result channel
`set spinconstraint`	Set target spins for spin constraints
`get spinconstraint`	Print target spins for spin constraints on result channel
`get atom spin`	Print atomic spins on result channel
`enable spinconstraint`	Switch on spin constraint
`disable spinconstraint`	Switch off spin constraint

6 Output

6.1 energy.dat: total energies

This file lists the total energies in each electronic step. The format for `scfDiag` is (7 columns in one long line)

```
<step> <accumulated time> <total energy> <free energy>
   <T=0 energy> <band energy> <entropy>
```

Units are Hartree, except for dimension-less entropy S^{el}.

The entropy arises from the partial occupations (Fermi-Dirac distribution or Methfessel-Paxton) used for metals. The connection between the different energies (E for energy, F for free energy) is

$$\begin{aligned} F(T) &= E(T) - k_\mathrm{B}T\,S^\mathrm{el}(T) \\ E(T=0) &\approx \frac{1}{2}[E(T)+F(T)] + \mathcal{O}(T^3) \end{aligned}$$

Here, $k_\mathrm{B}T$ is the electronic temperature parameter `ekt` parameter set in the Hamiltonian.

Note: Only the free energy is variational, and forces correspond to the derivatives of the free energy. Therefore, the Born-Oppenheimer surface at finite electronic temperature corresponds to the electronic free energy.

The format for `CCG` is

```
<step> <accumulated time> <total energy> <free energy>
```

Units are Hartree.

6.2 residue.dat: density residues

This file contains the density residue for each electronic step. The format is

```
<step> <residue>
```

For spin-polarized calculations, the third column lists the residue of the spin density.

6.3 eps.dat: eigenvalues

This file contains the computed eigenvalues. The format is

```
<ik> <eps_1> <eps_2> <eps_3> ...
```

Units are eV.

For spin-polarized calculations, the two spin channels have distinct eigenvalues contained separately in `eps.0.dat` and `eps.1.dat`, respectively.

6.4 rho.sxb: density (binary)

This file contains the final density in netcdf-format. The mesh order is z running fastest. For details on how to interpret the variables, please contact me.

6.5 waves.sxb: wave functions (binary)

This file contains the final wave functions in netcdf-format. It's pointless to try to process this file outside the SPHInX library. Several add-ons are available for processing the wavefunctions.

6.6 relaxedStr.sx: final atomic structure

This file list the final atomic positions in SPHInX format. If the optimization does not converge, it contains the last configuration calculated.

6.7 relaxHist.sx: geometry optimization history

This file lists for each calculated configuration the atomic positions and forces, in SPHInX format.

Each configuration corresponds to a structure format, with an additional item `force`, which lists the force vector in atomic units (Hartree/bohr).

Example:

```
structure  {
   cell = [[  10.0000,    0.0000,    0.0000],
           [   0.0000,   10.0000,    0.0000],
           [   0.0000,    0.0000,   10.0000]];
   species  {
      element="N";
      atom {coords = [   5.00000,    5.00000,    5.00000];
            force  = [  0.0027634,  0.0027634,  0.0027634]; }
   }
   species  {
      element="H";
      atom {coords = [   5.99349,    3.62616,    3.62616]; movable;
            force  = [-0.0191398,  0.0081741,  0.0081741]; }
      atom {coords = [   3.62616,    5.99349,    3.62616]; movable;
            force  = [  0.0081741,-0.0191398,  0.0081741]; }
      atom {coords = [   3.62616,    3.62616,    5.99349]; movable;
            force  = [  0.0081741,  0.0081741,-0.0191398]; }
   }
}
```

6.8 energy-structOpt.dat: geometry optimization energies

This file lists for each step the (electronic free) energy. The format is one step per line:

```
<it> <energy (in Hartree)>
```

The 0-th step is the initial structure.

6.9 fftwisdom.dat: FFTW plans

FFTW wisdom file. May be used to speed up FFTW planning in add-ons.

References

[1] S. Boeck, C. Freysoldt, A. Dick, L. Ismer, and J. Neugebauer, Comp. Phys. Commun. **182**, 543 (2011).

[2] C. Freysoldt, A. Mishra, M. Ashton, and J. Neugebauer, Physical Review B **102**, 045403 (2020).

[3] K. T. Tang, Physical Review **177**, 108 (1969).

[4] T. Gould and T. Bučko, J. Chem. Theory Computation **12**, 36033 (2016).

[5] M. Cococcioni and S. de Gironcoli, Physical Review B **71**, 035105 (2005).

[6] B. Lange, C. Freysoldt, and J. Neugebauer, Physical Review B **84**, 085101 (2011), URL `https://doi.org/10.1103%2Fphysrevb.84.085101`.

[7] J. Neugebauer and C. G. Van de Walle, in *Materials Theory, Simulations, and Parallel Algorithms, MRS Symposia Proceedings*, edited by E. Kaxiras, J. Joannopoulos, P. Vashisha, and R. K. Kalia (MRS, Pittsburgh, 1996), vol. 408.

[8] C. Freysoldt, S. Boeck, and J. Neugebauer, Phys. Rev. B **79**, 241103(R) (2009).

[9] C. Freysoldt, Comp. Mat. Sci. **133**, 71 (2017).

Index