
DFTB+ recipes

Release 11-09

B. Aradi

September 14, 2011

Contents

I	Setting up DFTB+	1
1	Compiling and installing	3
1.1	Prerequisites	3
1.2	Architecture dependent settings	3
1.3	Further customisation	4
1.4	Compiling the code	4
II	Basic usage	7
2	First calculation with DFTB+	9
2.1	Providing the input	9
2.2	Running DFTB+	13
2.3	Examining the output	14
3	Band structure, DOS and PDOS	21
3.1	Introduction	21
3.2	Creating the proper density	21
3.3	Plotting the density of states	23
3.4	Calculating the band structure	24

Part I

Setting up DFTB+

Compiling and installing

1.1 Prerequisites

In order to compile the code, the following components must be present on your system:

DFTB+ source The source code can be downloaded from the [DFTB+ website](#).

BLAS/LAPACK DFTB+ uses BLAS and LAPACK calls for its matrix manipulation operations. You need BLAS and LAPACK compatible libraries on your system.

Fortran 95 compiler You need a decent Fortran 95 compiler. DFTB+ relies pretty much on advanced Fortran 95 constructs. Some old compilers, not fully implementing the Fortran 95 standard, fail to compile the DFTB+. The compilers, for which a makefile is provided, should work. (Check out the release notes for the current release you are trying to compile.)

GNU Make In order to process the makefiles, you need GNU Make (version $\geq 3.79.1$). If there is no GNU Make on your system, you should download it from the [GNU make website](#) and compile it by yourself (which is pretty straightforward).

cpp The Fortran 95 source files of DFTB+ must be preprocessed with the standard C preprocessor.

awk The preprocessed files are postprocessed by the standard Unix tool *awk*.

Python (optional) Python is not needed for the compilation of the code, but for the testing of the resulting binary with the autotest system. You need Python version ≥ 2.3 for that.

1.2 Architecture dependent settings

First you have to unpack the downloaded source. For simplicity in this howto we assume that you unpacked the source in `/tmp`:

```
/tmp> tar xvzf ~/download/dftb-plus_1.2_src.tar.gz
/tmp> cd dftb+_1.2_src/
```

At first, you'll have to configure the make system to match the needs for your architecture and compiler. The system specific makefiles can be found in the subdirectory `sysmakes/`. If you don't find a makefile appropriate for your system, create your own by using `make.generic` as template. You should check and eventually modify the following variables in the system specific makefile:

FC90 Compiler to use. Despite the variable name, you have to provide a *Fortran 95* compiler.

FC90OPT Options to pass to the Fortran 95 compiler. You should set and tune the optimisation switches here.

CPP Name of the C preprocessor (usually `cpp`).

CPPOPT Options to pass to the C preprocessor. On some architectures the C preprocessor needs extra options in order to process files containing a non-C source code (e.g. the `-traditional` switch on linux). Additionally to these options, the debug level for the compilation must be passed to the preprocessor as `-DDEBUG=$(DEBUG)`.

CPPPOST Utility to use for postprocessing the files coming out from `cpp` (e.g. making sure, lines are not longer than 132 characters, removing debug info created by `cpp`, etc). You should set this variable to:

```
$(ROOT)/utils/fpp/fpp.sh <method>
```

where `<method>` can be one of the following keywords (depending on your compiler):

general The `#line` directives in the output file are kept. (Very convenient when debugging, but not all compilers can process source files with such directives.)

nocntln `#line` directives removed from continuation lines.

noln Does not add any `#line` directives when substituting CPP macros.

noln2 Removes any `#line` directives occurring in the preprocessed source.

LN Linker to use for linking the object files. (It is usually set to the same value as `FC90`.)

LNOPT Options to pass to the linker (e.g. `-static` if you want to create a statically linked binary).

LIBOPT Library related options for the linker. You should specify here the location of your LAPACK and BLAS libraries if they are not in the search path of the linker. If those libraries are for example in `/usr/local/lib`, you should set:

```
LIBOPT = -L/usr/local/lib
```

LIB_BLAS Linker option for link blas (e.g. `-lblas`)

LIB_LAPACK Linker option for linking lapack (e.g. `-llapack`).

Next you have to copy in the root of the source directory the file `Makefile.user.template` to `Makefile.user` and set the variable `$(ARCH)` to your architecture (to the suffix of your makefile in the subdirectory `sysmakes`). For example for `x86_64-linux-ifort` you would have to set:

```
# Set your architecture. Make sure, that 'sysmakes/make.$(ARCH)' exists for  
# you architecture. If not, create your own using 'sysmakes/make.generic'  
# as template.  
ARCH = x86_64-linux-ifort
```

1.3 Further customisation

You can do further customisation by overriding any variables used by the make system in `Makefile.user`. The following variables could be of interest: (The variable `$(ROOT)` points to the root of the DFTB+ source directory.)

OBJDIR_PARENT Directory, in which the subdirectory containing the object files and the compiled binary should be created. Per default it is set to the directory, where you issue the `make` command (working directory).

OBJDIR_SUFFIX Suffix to append in the directory name for the object files after `_obj`. Per default it is set to `verbl_$(ARCH)`.

INSTALLDIR The directory, where to copy the DFTB+ binary, when issuing `make install`. Defaults to the working directory.

PRGDFTB_TESTDIR Location of the DFTB+ autotest suite (see next section). The autotest suite must be download separately from the code. It is not needed for the compilation, only for testing the resulting binary. The default is `$(ROOT)/../autotest`.

PRGDFTB_TESTFILE File containing the tests to calculate. Default: `$(PRGDFTB_TESTDIR)/tests`.

1.4 Compiling the code

After having customised the make system for your architecture, enter the `prg_dftb/` directory. First you should make sure, that you don't have any remnants of previous compilations, so type:

```
make distclean
```

Then invoke make without any arguments:

```
make
```

You should see something like that:

```
make[1]: Entering directory `/tmp/dftb+_1.2/src/prg_dftb/_obj_x86_64-linux-ifort'
echo "" > _dependencies
for dep in ../../lib_common/Makefile.dep ../../lib_dftb/Makefile.dep ../../lib_g
eoopt/Makefile.dep ../../lib_io/Makefile.dep ../../lib_math/Makefile.dep ../../l
ib_md/Makefile.dep ../../lib_mixer/Makefile.dep ../../lib_type/Makefile.dep ../.
./includes/Makefile.dep ../../ext_xmlf90/Makefile.dep ../Makefile.dep ../../lib_
math/Makefile.local; do cpp -traditional -DDEBUG=0 $dep >> _dependencies; done
make[1]: Leaving directory `/tmp/dftb+_1.0/src/prg_dftb/_obj_x86_64-linux-ifort'
make[1]: Entering directory `/tmp/dftb+_1.0/src/prg_dftb/_obj_x86_64-linux-ifort'
cpp -traditional -DDEBUG=0 -I../../lib_common/ -I../../includes ../../lib_common
/allocate.F90 | ../../utils/fpp/fpp.sh nocntln > allocate.f90
ifort -O2 -xW -ip -o allocate.o -c allocate.f90
touch _mod_allocation
:
make[1]: Leaving directory `/tmp/dftb+_1.0/src/prg_dftb/_obj_x86_64-linux-ifort'
```

The resulting binary dftb+ can be found in the directory \$(OBJDIR_PARENT)/_obj\$(OBJDIR_SUFFIX). If you did not change those variables, you find it below prg_dftb/_obj_\$(ARCH), where \$(ARCH) contains the value, you set in Makefile.user. By issuing:

```
make install
```

the binary is copied to the current directory (prg_dftb) or to the place specified with the variable \$(INSTALLDIR) in Makefile.user.

Part II

Basic usage

First calculation with DFTB+

This chapter should serve as a tutorial guiding you through your first calculation with DFTB+. As an example, the equilibrium geometry of a water molecule is calculated. In order to execute the calculation described here, you will need the Slater-Koster set *mio*, which you can download from the DFTB.ORG (<http://www.dftb.org>).

The description here is based on DFTB+ version 1.2, the input/output files in later versions may slightly differ from those shown here.

2.1 Providing the input

At first, you have to create the input for the code. DFTB+ accepts either Human-readable Structured Data (HSD) or eXtended Markup Language (XML) input. In this tutorial HSD will be used, and in this case the input file must be called `dftb_in.hsd`. The input file used in this howto looks as follows:

```
Geometry = GenFormat {
3 C
  O H

  1 1  0.000000000000E+00 -0.100000000000E+01  0.000000000000E+00
  2 2  0.000000000000E+00  0.000000000000E+00  0.783064000000E+00
  3 2  0.000000000000E+00  0.000000000000E+00 -0.783064000000E+00
}

Driver = ConjugateGradient {
  MovedAtoms = 1:-1
  MaxForceComponent = 1E-4
  MaxSteps = 100
  OutputPrefix = "geom.out"
}

Hamiltonian = DFTB {
  SCC = Yes
  SlaterKosterFiles {
    O-O = "O-O.skf"
    O-H = "O-H.skf"
    H-O = "O-H.skf"
    H-H = "H-H.skf"
  }
  MaxAngularMomentum {
    O = "p"
    H = "s"
  }
  Filling = Fermi {
    Temperature [Kelvin] = 0.0
  }
}

Options {}

ParserOptions {
  ParserVersion = 4
}
```

The order of the specified blocks in the HSD input is arbitrary. You are free to capitalise the keywords and physical units as you like, since they are case-insensitive. This is not valid, however, for string values, especially if they are specifying file names.

2.1.1 Geometry

The `Geometry` block contains the types and coordinates of the atoms in your system. The geometry of the system in the sample input file is provided in the so called “gen” format, which was the traditional geometry input format of the DFTB method. The formal description of this format can be found in the DFTB+ manual. The current example:

```
Geometry = GenFormat {
  3 C           # 3 atoms, non-periodic cluster
  O H          # Two elements, 1 - O, 2 - H
  # Index Type Coordinates
    1 1 0.0000000000E+00 -0.1000000000E+01 0.0000000000E+00
    2 2 0.0000000000E+00 0.0000000000E+00 0.7830640000E+00
    3 2 0.0000000000E+00 0.0000000000E+00 -0.7830640000E+00
```

specifies a cluster system with 3 atoms of the type O and H. The coordinates of the atoms in the “gen” format are given in Angstroms. The first column of integers contains the sequential number of the atoms in the system (ignored by the parser). The second column contains the type of each atom, given as the position of the appropriate element in the element list in the second line. The `GenFormat{}` is not the only method to specify the geometry, you should check the manual for other methods.

As demonstrated above, it is possible to put arbitrary comments in the HSD input after a hashmark (#) character. Everything between it and the end of the line is ignored by the parser.

Very often, the geometry is stored in an external file different from `dftb_in.hsd`. To save you the copying and pasting from that file into the input file, you can use the file inclusion feature of the HSD format:

```
Geometry = GenFormat {
  <<< "input_geometry.gen"
}
```

The `<<<` operator includes the specified file as raw data. (The file is not checked for any HSD constructs.) In the example above, the file `input_geometry.gen` must be in gen format of course.

2.1.2 Driver

After having specified the geometry of your system, you should decide, what to do with that geometry. The `Driver` property determines, how the geometry should be changed (if at all) during the run. If you only would like to make a static calculation, you must set it to an empty value like:

```
Driver = {} # Empty value for the driver
```

In the current example:

```
# Do conjugate gradient optimisation
Driver = ConjugateGradient {
  MovedAtoms = 1:-1           # Move all atoms in the system
  MaxForceComponent = 1.0e-4   # Stop if maximal force below 1.0e-4
  MaxSteps = 100               # Stop after maximal 100 steps
  OutputPrefix = "geom.out"    # Final geometry in geom.out.{xyz,gen}
}
```

the molecule is relaxed using the conjugate gradient method. The entire range of atoms from the first (1) until and including the last one (-1) is allowed to move. Instead of `1:-1` you could also have written:

```
MovedAtoms = 1:3 # Atoms from the 1st until the 3rd
```

or:

```
MovedAtoms = O H # Select O and H atoms.
```

or:

```
MovedAtoms = 1 2 3 # Explicitely listing all atom numbers.
```

In our case the geometry optimisation continues as long as the maximal force component is bigger than $1e-4$ atomic units (Hartree/Bohr). Numeric values are per default interpreted in atomic units. The HSD format offers, however, the possibility to use alternative units by specifying the unit (modifier) of the value in square brackets before the equal sign. For example instead of the original specification, you could have used:

```
MaxForceComponent [eV/AA] = 5.14e-3 # Force in Electronvolt/Angstrom
```

or:

```
MaxForceComponent [Electronvolt/Angstrom] = 5.14e-3
```

The `MaxSteps` property specifies, after how many geometry optimisation steps the program should stop, even if the specified tolerance for the maximal force component could not be reached.

Finally, the `OutputPrefix` property specifies the name of the file containing the actual geometry during the optimisation (and the final geometry at the end of the calculation). The geometry is written in gen and xyz formats to the files obtained by appending ".gen" and ".xyz" suffixes to the specified name (`geom.out.gen` and `geom.out.xyz` in our case.) The *dptools* package on the [DFTB+ website](#) contains scripts to convert between the gen and the xyz formats (and various other formats).

2.1.3 Hamiltonian

In order to calculate the various properties of your system, you have to decide upon the way how the Hamiltonian for your system should be built. At the moment DFTB+ eases the decision quite a lot, since it only supports a Density Functional based Tight Binding Hamiltonian (with some extensions). In our example, the chosen self-consistent DFTB Hamiltonian has the following properties:

```
Hamiltonian = DFTB { # DFTB Hamiltonian
  SCC = Yes # Use self consistent charges
  SlaterKosterFiles { # Specifying Slater-Koster files
    O-O = "O-O.skf"
    O-H = "O-H.skf"
    H-O = "O-H.skf"
    H-H = "H-H.skf"
  }
  MaxAngularMomentum { # Maximal l-value of the various species
    O = "p"
    H = "s"
  }
  Filling = Fermi { # No electronic temperature
    Temperature [Kelvin] = 0.0
  }
}
```

In this example the SCC-DFTB method is used for building up the Hamiltonian (and calculating the total energy, forces, etc.). This method considers the charge transfer between the atoms. In order to find the final charges, it has to make a few iterations, until the charges are converged. Convergence is reached if the difference between the charges used to build the Hamiltonian and the charges obtained after the diagonalisation of the Hamiltonian is below a certain tolerance. (The default is $1e-4$, but can be tuned with the `SCCTolerance` option.) If the convergence is not reached within a certain number of iterations, the code calculates the total energy using the charges obtained so far and stops with an appropriate warning message. (Number of the maximal scc-iterations is per default 100, and can be tuned via the `MaxSCCIterations` option.)

The integral tables (together with other atomic and diatomic parameters) necessary for building the Hamiltonian are stored in the so called Slater-Koster files. Those files always describe the interaction between atom pairs. Therefore, you have to specify for every atom pair the corresponding Slater-Koster file:

```
SlaterKosterFiles = {                                # Specifying Slater-Koster files
  O-O = "O-O.skf"
  O-H = "O-H.skf"
  H-O = "O-H.skf"
  H-H = "H-H.skf"
}
```

If you use a certain file naming convention, you can avoid typing all the file names by specifying only the generating pattern. The input:

```
SlaterKosterFiles = Type2FileNames { # File names with two atom type names
  Prefix = "" # No prefix before first type name
  Separator = "-" # Dash between type names
  Suffix = ".skf" # Suffix after second type name
}
```

would generate exactly the same file names as in the example above. If the Slater-Koster files are in a different directory as the input file `verblftb_in.hsdl`, you can append the path as prefix:

```
SlaterKosterFiles = Type2FileNames { # File names from two atom type names
  Prefix = "/home/aradi/slako/mio-0-1" # Path as prefix
  Separator = "-" # Dash between type names
  Suffix = ".skf" # Suffix after second type name
}
```

The historical Slater-Koster file format does not contain any information about the orbitals, which were considered when generating the interaction tables. Therefore, you must provide the highest angular momentum for every element as *s*, *p*, *d* or *f*. This information can be obtained from the documentation of the Slater-Koster files. In the distributed standardised sets (as distributed on <http://www.dftb.org>) this information is contained in the documentation appended to the end of each SK-file.

The systems are assumed to be neutral. If you'd like to calculate charged systems, you have to use the `Charge` option. Similarly, the system is assumed to be spin-unpolarized. You can use, however, the option `SpinPolarisation` to change the standard behaviour.

The `Filling` option describes the method to use for filling up the one electron levels with electrons. Here the Fermi-Dirac statistics is used. The filling functions usually require further parameters (e.g the temperature).

2.1.4 Options

The `Options` block contains a few global settings for the code. In the current example, no options are specified. You could have even leave out the:

```
Options {}
```

line in the input, since the default value for the `Options` block is the empty block.

2.1.5 ParserOptions

This block contains options which are interpreted by the parser itself and are not passed to the main program. The most important of those options is the `ParserVersion` option, which tells the parser, for which version of the parser the current input file was created for. If this is not the current parser but an older one, the parser internally automatically converts the old input to the new format.

The version number of the parser in the current DFTB+ code is always printed out at the program start. It is a good habit to set this value in your input files explicitly, like in our case:

```
ParserVersion = 4
```

It allows you to use your input file with the future versions of DFTB+ without adapting it by hand, if the input format changed in the more recent versions.

2.2 Running DFTB+

After creating the main input file, you should make sure that all the other needed files (Slater-Koster files, targets of eventual file inclusions in the HSD input) are at the right place. In our case, only the Slater-Koster files needs to be present, and since we specified them without path, they must be in the same directory as `dftb_in.hsd` itself. This howto uses Slater-Koster files from the `mio-0-1 SK-set`.

In order to make the calculation, you should invoke DFTB+ without any arguments in the directory containing the file `dftb_in.hsd`:

```
dftb+
```

Assuming the binary `dftb+` lies in your search path, you should obtain an output starting with:

```
=====
==
== Density Functional based Tight Binding with a lot of extensions (DFTB+)
==
== Release: 1.2 (p0)
==
== (ParserVersion = 4)
==
=====

*****
** Parsing and initializing
*****

Interpreting input file 'dftb_in.hsd'
-----

Processed input in HSD format written to 'dftb_pin.hsd'

Starting initialization...
-----
Mode: Conjugate gradient relaxation
Self consistent charges: Yes
SCC-tolerance: 0.100000E-04
Max. scc iterations: 1000
Spin polarisation: No
Nr. of up electrons: 4.000000
Nr. of down electrons: 4.000000
Periodic boundaries: No
Diagonalizer: Divide and Conquer
Mixer: Broyden mixer
Mixing parameter: 0.200000
Maximal SCC-cycles: 1000
Nr. of chrg. vec. in memory: 1000
Nr. of moved atoms: 3
Max. nr. of geometry steps: 100
Force tolerance: 0.100000E-03
Electronic temperature: 0.100000E-07
Initial charges: Set to neutral
Maximal angular momentum: O: P
H: S

Extra options:
-----

*****
** Geometry step: 0
```

```

*****
iSCC Total electronic   Diff electronic   SCC error
  1  -0.39511797E+01    0.00000000E+00    0.88081627E+00
  2  -0.39705438E+01   -0.19364070E-01    0.55742893E+00
  3  -0.39841371E+01   -0.13593374E-01    0.32497352E-01
  4  -0.39841854E+01   -0.48242063E-04    0.19288772E-02
  5  -0.39841856E+01   -0.17020682E-06    0.87062163E-05
>> Charges saved for restart in charges.bin

Total Energy:                -3.979879
Total Mermin free energy:     -3.979879
Maximal force component:      0.187090

```

```

*****
** Geometry step:    1
*****

iSCC Total electronic   Diff electronic   SCC error
  1  -0.40495557E+01    0.00000000E+00    0.92334379E-01
.
.
.

```

If this is the case, you managed to run DFTB+ for the first time. Congratulation!

2.3 Examining the output

DFTB+ communicates through two channels with the you: by printing information on the standard output (which you should probably redirect into a file to keep for later evaluation) and by writing information to various files. In the following, the most important of those should be introduced and analysed.

2.3.1 Standard output

The first thing appearing on standard output after the start of DFTB+ is the program header:

```

=====
==
== Density Functional based Tight Binding with a lot of extensions (DFTB+)
==
== Release: 1.2 (p0)
==
== (ParserVersion = 4)
==
=====

```

This tells you which program you are using (DFTB+), which release (1.2) and which patchlevel (p0). Then, you get the version of the parser used in this DFTB+ release.

As already discussed before, it is a good habit to indicate this version number explicitly in your input in the `ParserOptions` block, like that:

```

ParserOptions {
  ParserVersion = 4
}

```

Then the parser starts to interpret your input, reads in the necessary SK-files and writes the full input to `dftb_pin.hsd`:

```
*****
** Parsing and initializing
*****
```

Interpreting input file 'dftb_in.hsd'

Reading SK-files:

```
O-O.skf
O-H.skf
O-H.skf
H-H.skf
```

Done.

Processed input in HSD format written to 'dftb_pin.hsd'

You do not have to explicitly set all the possible options for DFTB+ in the input, as for most of them there are default values set by the parser, if the according specification is missing in the input. If you want to know what default values had been set for those missing specifications, you should look at the processed input file `dftb_pin.hsd`, which contains the values for all the possible input settings. (See next subsection.)

After that, the DFTB+ code is initialised, and for the most important quantities the provided values are printed out:

Starting initialization...

```
-----
Mode:                               Conjugate gradient relaxation
Self consistent charges:            Yes
SCC-tolerance:                       0.100000E-04
Max. scc iterations:                 100
Ewald alpha parameter:              0.000000E+00
Spin polarisation:                  No
Nr. of up electrons:                 4.000000
Nr. of down electrons:               4.000000
Periodic boundaries:                 No
Diagonalizer:                       Divide and Conquer
Mixer:                               Broyden mixer
Mixing parameter:                    0.200000
Maximal SCC-cycles:                  100
Nr. of chrg. vec. in memory:         100
Nr. of moved atoms:                  3
Max. nr. of geometry steps:          100
Force tolerance:                     0.100000E-03
Electronic temperature:              0.100000E-07
Initial charges:                     Set automatically (system chrg: 0.000E+00)
Included shells:                     O: s, p
                                       H: s
```

Extra options:

As you can see, all quantities (e.g. force tolerance, electronic temperature) are converted to the internal units of DFTB+, namely atomic units (with Hartree as the energy unit).

Then the program starts:

```
*****
** Geometry step: 0
*****
```

iSCC	Total electronic	Diff electronic	SCC error
1	-0.39511797E+01	0.00000000E+00	0.88081627E+00
2	-0.39705438E+01	-0.19364070E-01	0.55742893E+00
3	-0.39841371E+01	-0.13593374E-01	0.32497352E-01
4	-0.39841854E+01	-0.48242063E-04	0.19288772E-02

```

5    -0.39841856E+01   -0.17020682E-06    0.87062163E-05

Total Energy:                -3.9798793068 H
Total Mermin free energy:    -3.9798793068 H
Maximal force component:     0.187090E+00
>> Charges saved for restart in charges.bin
:
```

Since this calculation is an SCC calculation, DFTB+ has to iterate the charges until the specified convergence criteria is fulfilled. In every cycle, you get information about the value of the electronic energy, the difference to the value in the previous SCC cycle, and the error in the charges for which the tolerance value can be specified in the input (SCCTolerance).

If the SCC cycle converged, the total energy including SCC and repulsive contributions is calculated, and similarly the total free energy. Additionally the biggest force component in the system is indicated.

Then the driver changes the geometry of the system, and the same calculation as before is executed for the new geometry. This is done, as long as the geometry does not converge:

```

*****
** Geometry step: 12
*****

iSCC Total electronic   Diff electronic         SCC error
1    -0.41505783E+01    0.00000000E+00          0.20093314E-02
2    -0.41505783E+01   -0.21634570E-07          0.14891915E-02
3    -0.41505784E+01   -0.26364940E-07          0.27059012E-07

Total Energy:                -4.0779379339 H
Total Mermin free energy:    -4.0779379339 H
Maximal force component:     0.281596E-05
>> Charges saved for restart in charges.bin
```

Geometry converged

If the geometry does not converge before the number of maximal geometry steps is reached, you will get an appropriate warning. Assuming the `MaxSteps` option had been set to 6 in the input, you would obtain:

```

*****
** Geometry step: 6
*****

iSCC Total electronic   Diff electronic         SCC error
1    -0.41414787E+01    0.00000000E+00          0.12689108E-01
2    -0.41414797E+01   -0.96455051E-06          0.93470428E-02
3    -0.41414808E+01   -0.11439438E-05          0.17369905E-05

Total Energy:                -4.0774100809 H
Total Mermin free energy:    -4.0774100809 H
Maximal force component:     0.207932E-01
>> Charges saved for restart in charges.bin
WARNING!
-> !!! Geometry did NOT converge!
```

2.3.2 dftb_pin.hsd

As already mentioned, the processed input file `dftb_pin.hsd` is an input file generated from your input in `dftb_in.hsd` by setting the default values for all unspecified options and by converting some of the input quantities to atomic units. For example, in our case in the `ConjugateGradient` block several unspecified options would appear, for which sensible default values had been set:

```

Driver = ConjugateGradient {
  MovedAtoms = 1:-1
  MaxForceComponent = 1E-4
  MaxSteps = 100
  OutputPrefix = "geom.out"
  LatticeOpt = No
  AppendGeometries = No
  ConvergentForcesOnly = Yes
  Constraints = {}
}

```

Similarly, in the `DFTB{}` block the switch for the orbital resolved SCC, for example, had been set to the default value of `No`:

```
OrbitalResolvedSCC = No
```

The options, which had been set explicitly set in the input, are unchanged (eventually converted to atomic units). The file `dftb_pin.hsd` is a valid HSD input file, you can use it as input (after renaming it to `dftb_in.hsd`). It has always the format suitable for the current parser, even if the input in `dftb_in.hsd` was in an older format (indicated by the appropriate `ParserVersion` option). Therefore, the `ParserVersion` option in the processed input file `dftb_pin.hsd` is always set to the version of the current parser.

2.3.3 detailed.out

This file contains detailed informations about the properties of your system, as calculated during the last SCC cycle. It is updated continuously during the run. All the numerical values in it mean atomic units unless explicitly specified otherwise.

It contains (among others) the number of the last geometry step, a summary about the last SCC cycle and coordinates of the moved atoms:

```
Geometry optimization step: 12
```

```

*****
iSCC Total electronic   Diff electronic       SCC error
  3   -0.41505784E+01   -0.26364940E-07    0.27059012E-07
*****

Coordinates of moved atoms (au):
  1   0.00000000   -1.35305500    0.00000000
  2   0.00000000   -0.26833549    1.47114130
  3   0.00000000   -0.26833549   -1.47114130

```

Then the net atomic charges for each atom follow (in case of H₂O showing a strong electron transfer from the hydrogens to the oxygen):

```

Net atomic charges (e)
Atom      Net charge
  1      -0.59261123
  2       0.29630561
  3       0.29630561

```

Then the eigenvalues in both Hartree and electronvolt and the occupations of the individual levels for all the possible spin states. For spin unpolarised calculations (like this one) you get only one value, since the eigenlevels are twofold degenerated:

```

Eigenvalues /H
 -0.84898563
 -0.41433519
 -0.31375652
 -0.25917554

```

```
0.39924777
0.55837394
```

```
Eigenvalues /eV
-23.10207437
-11.27463412
-8.53774939
-7.05252519
10.86408466
15.19412800
```

```
Fillings
2.00000
2.00000
2.00000
2.00000
0.00000
0.00000
```

In a collinearly spin polarised calculation you would obtain separate values for spin up and spin down.

Then you obtain the number of total electrons in the system, and the number of electrons on each atom, each l-shell (s, p, d, etc.) and each orbital (s, px, py, pz, ...) as calculated by Mulliken-analysis:

```
Nr. of electrons (up):      8.00000000
```

```
Atom populations (up)
Atom      Population
  1         6.59261123
  2         0.70369439
  3         0.70369439
```

```
l-shell populations (up)
Atom Sh.  l      Population
  1   1   0      1.73422185
  1   2   1      4.85838938
  2   1   0      0.70369439
  3   1   0      0.70369439
```

```
Orbital populations (up)
Atom Sh.  l  m      Population
  1   1   0   0      1.73422185
  1   2   1  -1      1.68106711
  1   2   1   0      1.17732227
  1   2   1   1      2.00000000
  2   1   0   0      0.70369439
  3   1   0   0      0.70369439
```

In our case due to the electronegativity difference the hydrogen atoms are positively polarised having only 0.704 electrons, while the oxygen is negatively polarised having 6.59 electrons (instead of the neutral state of 6 electrons).

After that you find the Fermi energy, the different energy contributions to the total energy and the total energy in Hartree and electronvolt units. If you are calculating at finite temperature, you should consider the Mermin free energy instead of the total energy:

```
Fermi energy:      -0.1453058436 H          -3.9540 eV
Band energy:      -3.6725057531 H        -99.9340 eV
TS:               0.0000000000 H          0.0000 eV
Band free energy (E-TS): -3.6725057531 H        -99.9340 eV
Extrapolated E(0K): -3.6725057531 H        -99.9340 eV
Input/Output electrons (q): 8.00000000    8.00000000

Energy H0:        -4.1689401713 H          -113.4426 eV
Energy SCC:       0.0183617979 H           0.4996 eV
```

```

Total Electronic energy:      -4.1505783734 H          -112.9430 eV
Repulsive energy:            0.0726404395 H           1.9766 eV
Total energy:                -4.0779379339 H          -110.9663 eV
Total Mermin free energy:    -4.0779379339 H          -110.9663 eV

```

Between the two energy blocks, the input and output charges of the last Hamiltonian diagonalisation are shown, so that you can check, that no charges got lost during the calculation.

Then you find the confirmation, that the SCC convergence had been reached in the last geometry step:

```
SCC converged
```

You should always make sure that this is true, so that the properties of your system were calculated by using convergent charges. Values obtained by using non convergent charges are usually meaningless.

Finally you get the forces on the atoms in your system. You get also the maximal force component occuring in your system and the maximal force occuring among the moved atoms. After the dipole moment in atomic units and Debye, you will see the confirmation that the geometry optimisation has reached convergence, so that all force components on the moved atoms are below the specified tolerance:

```
Full geometry written in geom.out.{xyz|gen}
```

```

Total Forces
-3.588084523591196E-029  7.229005349884687E-008  -1.768030166715562E-014
 2.187779435721165E-016  -3.614502072646353E-008  -2.815955031099104E-006
-2.187779435720806E-016  -3.614503277238335E-008  2.815955048779406E-006

```

```

Maximal derivative component:      0.281596E-05 au
Max force for moved atoms::        0.281596E-05 au

```

```

Dipole moment   :    0.00000000    0.64281696    0.00000000 au
Dipole moment   :    0.00000000    1.63387787    0.00000000 Debye

```

```
Geometry converged
```

As indicated above, the final geometries can be found in xyz and gen formats in the output files `geom.out.xyz` and `geom.out.gen` in the current case. (The package *dptools*, which can be downloaded from the [DFTB+ website](#) contains some scripts to convert between xyz, gen and other geometry formats.)

2.3.4 band.out

For big systems, and especially for periodic systems with many k-points, it can become quite difficult to get a good overview over the one electron levels and their occupations in `detailed.out`. Therefore, an extra file `band.out` is created, which contains this information in a more human readable format:

```

KPT          1  SPIN          1  KWEIGHT  1.000000000000000
  -23.10207    2.00000
  -11.27463    2.00000
   -8.53775    2.00000
   -7.05253    2.00000
   10.86408    0.00000
   15.19413    0.00000

```

The eigenenergies are in atomic units. You can use the scripts `dp_bands` in the *dptools* package to convert the data in `band.out` to NXY-format, which can be visualized with common 2D plotting tools.

Despite its name, the file `band.out` is created also for non-periodic systems, containing the eigenenergies and occupation numbers for the calculated molecule. (You should ignore the k-point index and the k-point weight in the first line for that case.)

2.3.5 results.tag

If you want to process the results of DFTB+ with another program, you should not extract the informations from the standard output or the human readable output files (`detailed.out`, `band.out`, etc.), since their format could significantly change between subsequent releases of DFTB+. By setting the `WriteResultsTag` to `Yes` in the `Options {}` block:

```
Options {  
  WriteResultsTag = Yes  
}
```

you obtain the file `results.tag` at the end of your calculation containing some of the most important data in a format easily parsed by a script or a program. The file contains entries like:

```
forces                :real:2:3,3  
-0.277549616145533E-028  0.722900585226061E-007 -0.427713420236842E-013  
-0.109618479408687E-015 -0.361450127051022E-007 -0.281595502166221E-005  
 0.109618479408715E-015 -0.361450458036261E-007  0.281595506446131E-005
```

In the first line the name of the quantity is given, followed by its type (`real`, `integer`, `logical`). Then the rank of the quantity is given (0: scalar, 1: vector, 2: rank 2 matrix, etc.), followed by the size of each dimension. In the following the value of the given quantity is dumped in a free format.

2.3.6 Other output files

There are also other output files not discussed in detail here. They are only created, if the appropriate option in the `Options` block is set. Please consult the manual for further details.

Band structure, DOS and PDOS

This chapter demonstrates on the example of anatase (TiO₂) how the band structure, the density of states (DOS) and the partial density of states (PDOS) of periodic systems (wires, surfaces, solids) can be obtained using DFTB+.

The conversion scripts used here are part of the *dftools* package, which can be obtained from the [DFTB+ website](#). In order to execute the calculation, you will need the Slater-Koster sets `mio` and `tiorg`. The sample input files in this chapter assume that the necessary Slater-Koster files had been copied into a subdirectory `mio_tiorg` below the directory with the `dftb_in.hsd` input file.

The description here is based on DFTB+ version 1.2, the input/output files in later versions may slightly differ from those shown here.

3.1 Introduction

The calculation of the band structure for a periodic system consists of two steps. First, the charges in the system must be calculated using a converged k-point sampling. Then, keeping the obtained charges fixed, the one-electron levels must be calculated for k-points chosen along specific lines for which the band structure should be determined.

3.2 Creating the proper density

In order to calculate a band structure in Density Functional Theory (DFT), at first the ground-state density for the given system must be obtained. In the DFTB picture that corresponds to obtaining the self-consistent charges of the atoms. The charges must be convergent with respect to two quantities in order to give correct results:

- Tolerance of the SCC cycle and
- quality of the k-point sampling.

In the current tutorial, the SCC tolerance is set to $1e-5$. For the k-point sampling the 8x8x8 Monkhorst-Pack set will be used. Both quantities ensure good convergence in the charges for anatase.

We will use the calculation with the proper density to obtain information about the density of states (DOS) and the partial density of states (PDOS) in anatase. Those information only make sense, when extracted from a system with a good k-point sampling. The input `dftb_in.hsd` could look like follows:

```
Geometry = GenFormat {
  6 F
  Ti O
  1 1 0.4393045491E-02 -0.4394122690E-02 -0.4185505032E-06
  2 1 -0.2456050838E+00 -0.7543932244E+00 0.5000007729E+00
  3 2 0.1997217007E+00 0.2106836749E+00 -0.1813953963E-02
  4 2 -0.4625010039E+00 0.4843137675E-01 0.4981557672E+00
  5 2 -0.2106822274E+00 -0.1997223911E+00 0.1816384188E-02
  6 2 -0.4843281768E-01 -0.5374990457E+00 0.5018414482E+00
  0.0000000000E+00 0.0000000000E+00 0.0000000000E+00
  -0.1903471721E+01 0.1903471721E+01 0.4864738245E+01
  0.1903471721E+01 -0.1903471721E+01 0.4864738245E+01
  0.1903471721E+01 0.1903471721E+01 -0.4864738245E+01
}
```

```
Hamiltonian = DFTB {
  SCC = Yes
  SCCTolerance = 1e-5
  SlaterKosterFiles = Type2FileNames {
    Prefix = "./mio_tiorg/"
    Separator = "-"
    Suffix = ".skf"
  }
  MaxAngularMomentum {
    Ti = "d"
    O = "p"
  }
  Filling = Fermi {
    Temperature [Kelvin] = 0.0
  }
  KPointsAndWeights = SupercellFolding {
    4 0 0
    0 4 0
    0 0 4
    0.5 0.5 0.5
  }
}

Analysis {
  ProjectStates {
    Region {
      Atoms = Ti
      ShellResolved = Yes
      Label = "dos_ti"
    }
    Region {
      Atoms = O
      ShellResolved = Yes
      Label = "dos_o"
    }
  }
}

ParserOptions {
  ParserVersion = 4
}
```

In the input above, the coordinates have been specified in relative (fractional) coordinates, which express the positions of the atoms as a linear combination of the lattice vectors. This is indicated by using the letter **F** in the first line of the geometry specification:

```
Geometry = GenFormat {
  6 F
  :
```

The k-points had been generated automatically using the `SupercellFolding` method, which enables among others the generation of Monkhorst-Pack schemes. In the current example, a k-point set equivalent to the Monkhorst-Pack scheme 4x4x4 had been generated. (For details how to specify the coefficients and the shift vectors, please consult the manual.):

```
KPointsAndWeights = SupercellFolding {
  4 0 0
  0 4 0
  0 0 4
  0.5 0.5 0.5
}
```

You can check by generating bigger k-point sets, that the current set gives an accuracy in the range of $1e-3$ eV. Also, by specifying a smaller SCC tolerance as the specified one ($1e-5$), you can check, that converging the charges more does not decrease the total energy significantly.

The DOS we will plot using the output in the file `band.out`. In order to obtain PDOS as well, the appropriate atoms have been specified, for which the density of states information should be stored in separate files. This can be done in the `Analysis` block using the `ProjectRanges` options. In our example:

```
Analysis {
  ProjectStates {
    Region {
      Atoms = Ti
      ShellResolved = Yes
      Label = "dos_ti"
    }
    Region {
      Atoms = O
      ShellResolved = Yes
      Label = "dos_o"
    }
  }
}
```

we decided to get the PDOS for the Ti and the O atoms separately. Each `Region` block specifies the atoms (either selected by specie or atomic ranges or as a combination of both), for which PDOS should be created. Additionally, you can select, whether you would like to see each angular momentum (s, p, d, etc.) separately or just summed up for the atom. With the `Label` tag you can specify the prefix for the data files created. Using the settings above, we will obtain 5 files: `dos_ti.1.dat`, `dos_ti.2.dat`, `dos_ti.3.dat`, `dos_o.1.dat` and `dos_o.2.dat`. The first three contain the PDOS for the s, p, and d shells of Ti, while the last two the same for the oxygen s and p shells.

3.3 Plotting the density of states

You can use the `dp_dos` program from the `dptools` package to take the eigenlevels stored in `band.out`, apply a gaussian smearing to them, and to store the result in a format, which can be easily plotted by any 2D visualization tool. You have to issue:

```
dp_dos band.out dos_total.dat
```

This would create a file `dos_total.dat` in NXY format, with the energies as X-values and the calculated DOS values as Y-values. You can tune the output by setting different options for `dp_dos`. Invoke it with the help option:

```
dp_dos -h
```

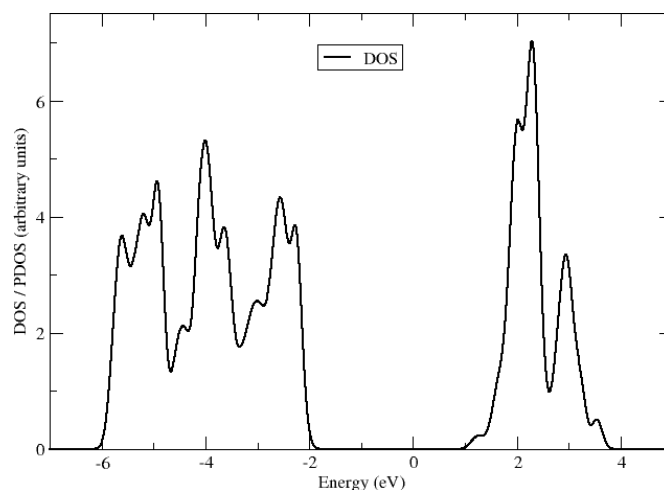
to obtain a detailed information about possible options. Visualizing the results with `xmgrace` for example by issuing:

```
xmgrace -nxy dos_total.dat
```

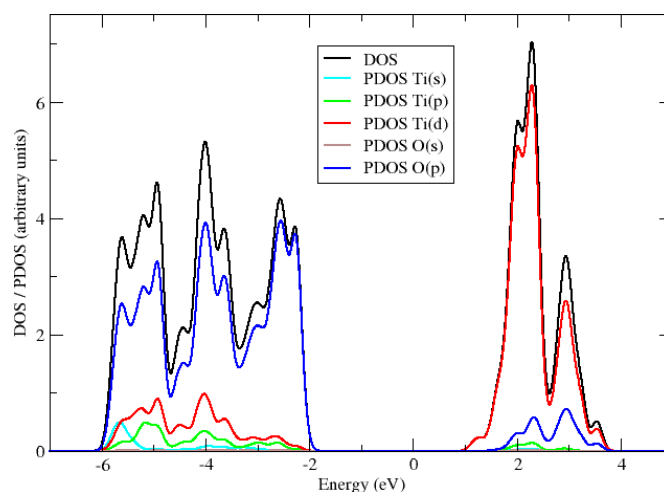
and by zooming into the region around the Fermi-level (showing the valence band edge and the conduction band edge), you should obtain a picture like this:

In order to investigate, from which states the valence band edge and the conduction are made of, we will visualize the contribution of the individual atomic shells to the band edges. For that, we have to convert the PDOS-files to NXY files. In the case of `dos_ti.1.dat` you could execute:

```
dp_dos -w dos_ti.1.out dos_ti.s.dat
```



and similar to all other PDOS files. It is important, that you specify the weighting option `-w` for the PDOS files, otherwise the total DOS (instead of the appropriate PDOS) will be created. By visualizing the obtained data files together with the total DOS, you should obtain a picture like



There you can see, that the valence band edge of anatase is entirely composed of the oxygen p-orbitals, while the conduction band edge is made of the d-orbitals of titanium.

3.4 Calculating the band structure

Once well converged charges for a system had been obtained, the band structure can be calculated at any chosen k-points. In our case, we will choose the Z-Gamma-X-P line. In order to do that, the following input will be used:

```
Geometry = GenFormat {
  6 F
  Ti O
  1 1 0.4393045491E-02 -0.4394122690E-02 -0.4185505032E-06
  2 1 -0.2456050838E+00 -0.7543932244E+00 0.5000007729E+00
  3 2 0.1997217007E+00 0.2106836749E+00 -0.1813953963E-02
  4 2 -0.4625010039E+00 0.4843137675E-01 0.4981557672E+00
  5 2 -0.2106822274E+00 -0.1997223911E+00 0.1816384188E-02
  6 2 -0.4843281768E-01 -0.5374990457E+00 0.5018414482E+00
  0.0000000000E+00 0.0000000000E+00 0.0000000000E+00
  -0.1903471721E+01 0.1903471721E+01 0.4864738245E+01
  0.1903471721E+01 -0.1903471721E+01 0.4864738245E+01
```

```

    0.1903471721E+01    0.1903471721E+01    -0.4864738245E+01
}

Hamiltonian = DFTB {
  SCC = Yes
  ReadInitialCharges = Yes
  MaxSCCIterations = 1
  SlaterKosterFiles = Type2FileNames {
    Prefix = "./mio_tiorg/"
    Separator = "-"
    Suffix = ".skf"
  }
  MaxAngularMomentum {
    Ti = "d"
    O = "p"
  }
  Filling = Fermi {
    Temperature [Kelvin] = 0.0
  }
  KPointsAndWeights = Klines {
    1  0.5  0.5  -0.5  # Z
    20 0.0  0.0  0.0  # G
    45 0.0  0.0  0.5  # X
    10 0.25 0.25 0.25 # P
  }
}

ParserOptions {
  ParserVersion = 4
}

```

The input is (must be) practically the same as in the previous case, with only a few adaptations:

- If a `Driver` was used to get the final geometry, this must be disabled as there should be no relaxation during the band structure calculation.
- As we want to use the charge density obtained in the previous calculation for well converged charges, you have to copy the `charges.bin` file from the previous calculation to the directory of the current calculation. At the same time, you must instruct the code to read those charge densities, by setting:

```
ReadInitialCharges = Yes
```

- Since we want to use the well converged charges to obtain the band structures and do not want to change them during the calculation, the maximal number of SCC cycles should be set to 1:

```
MaxSCCIterations = 1
```

- Finally, the k-points should be adapted according to the lines in the Brillouin-zone, along which you wish to obtain the band structure. You can achieve that by using the `Klines` directive:

```

KPointsAndWeights = Klines {
  1  0.5  0.5  -0.5  # Z
  20 0.0  0.0  0.0  # G
  45 0.0  0.0  0.5  # X
  10 0.25 0.25 0.25 # P
}

```

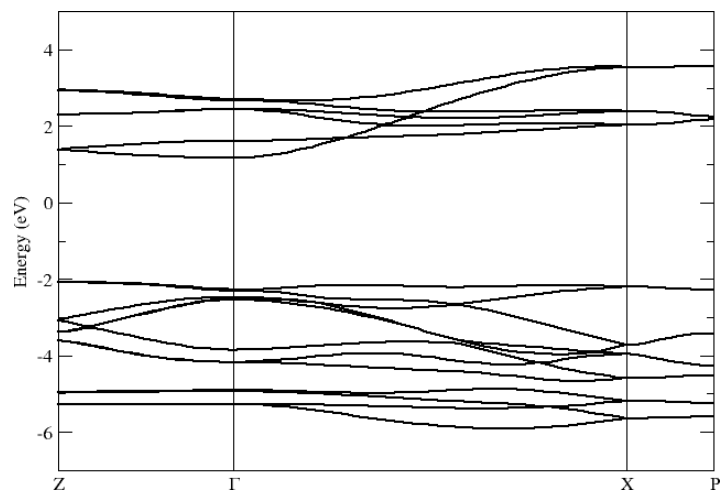
Every line specifies a line segment. The first column gives the number of k-points along the line segment between (but excluding) the end of the previous line segment and the k-point specified in the next three columns (which is the end point of the current line segment). The specified number of k-points is evenly distributed along the line, the last k-point coincides with the end point of the segment. The coordinates of the k-points are fractional coordinates (given in the coordinate system spawned by the reciprocal lattice vectors of the periodic structures).

The starting point of the first line segment is the Gamma point, but you can override it by setting a line segment with one point only as demonstrated above for the Z-point.

Running DFTB+ with the input above, the eigenlevel spectrum is calculated at the required k-points. The results are written to the file `detailed.out` and in more readable format to `band.out`. You can use the script `dp_bands` from the *dptools* package to convert it into NXY format. By issuing:

```
dp_bands band.out band
```

you would obtain a file `band_tot.dat` containing the band structures. After visualizing it, you should see something like below.



Note, DFTB+ enumerates the k-points along the lines you specified starting by one. The vertical bars corresponding to the special points Z, Gamma, X und P must be therefore inserted on positions 1, 21, 66, 76.