DFTB+

Version 24.1

USER MANUAL

Contents

Chapter 1

Introduction

The code DFTB+ is the Fortran 2003 successor of the old DFTB code, which implements the density functional based tight binding approach [25]. The code has been completely rewritten from scratch and extended with various features. The most recent features of the code are described in Ref. [37].

The main features of DFTB+ are:

- Non-scc and scc calculations (with an expanded range of SCC accelerators)
  - Cluster/molecular systems
  - Periodic systems (arbitrary k-point sampling, band structure calculations, etc.)
  - Open boundary conditions (wire and semi-infinite contacts)
- l-shell resolved calculations possible
- Spin polarised calculations (both collinear and non-collinear spin)
- Geometry and lattice optimisation
- Geometry optimisation with constraints (in xyz-coordinates)
- Molecular dynamics (NVE, NPH, NVT and NPT ensembles as well as metadynamics)
- Numerical vibrational mode calculations
- Finite temperature calculations
- Dispersion corrections (van der Waals interactions)
- Ability to treat f-electrons
- Linear response excited state calculations for cluster/molecular systems
- Geometry optimisation and molecular dynamics in excited states
- LDA+U/pSIC extensions
- $L \cdot S$ coupling
- 3rd order on-site corrections (improved H-bonding)

- Long range hybrid corrections
- REKS calculations for a strongly correlated system
- QM/MM coupling with external point charges (smoothing possible)
- GPU acceleration and OpenMP and MPI parallelisation with a range of electronic solvers
- Electronic transport calculations (non-equilibrium Green function methodology, transmission calculations)
- More general electrostatic boundary conditions via a Poisson equation electrostatic solver
- Automatic code validation (autotest system)
- New user friendly, extensible input format (HSD)
- Dynamic memory allocation
- Additional tool for generating cube files for charge distribution, molecular orbitals, etc. (Waveplot)
- Excitation energies for cluster/molecular systems using the particle-particle Random Phase Approximation.

The recent releases of the code are listed in appendix A.

Chapter 2

Input for DFTB+

DFTB+ can read the Human-friendly Structured Data format (HSD). If you are not familiar with the HSD format, a detailed description is given in appendix B. The input file for DFTB+ must be named `dftb_in.hsd` and must be present in the working directory. After processing the input, DFTB+ creates a file of the parsed input called `dftb_pin.hsd`. This contains the user input as well as default values for unspecified options. The file also contains the version number of the current input parser. You should always keep this file, since if you want to exactly repeat your calculation with a later version of DFTB+, it is recommended to use this file instead of the original input. (You must of course rename `dftb_pin.hsd` into `dftb_in.hsd`.) This guarantees that you will obtain the same results, even if the defaults for some non specified options have been changed in the meanwhile (as `dftb_pin.hsd` stores the original choice).

The following sections list properties and options that can be set in DFTB+ input. The first column of each of the tables of options specifies the name of a property. The second column indicates the type of the expected value for that property. The letters "l", "i", "r", "s", "p", "m" stand for logical, integer, real, string, property list and method type, respectively. An optional prefixing number specifies how often (if more than once) this type must occur. An appended "+" indicates arbitrary occurrence greater than zero, while "*" allows also for zero occurrence. Alternative types are separated by "|". Parentheses serve only to delimit groups of settings.

Sometimes a property is only interpreted on meeting some condition(s). If this is the case, the third column gives details of the requirement(s). The fourth column contains the default value for the property. If no default value is specified ("-"), the user is required to assign a value to that property. The description of the properties immediately follows the table. If there is also a more detailed description available for a given keyword somewhere else, the appropriate page number appears in the last column.

Some properties are allowed to carry a modifier to alter the provided value (e.g. converting between units). The possible modifiers are listed between brackets ([]) in the detailed description of the property. If the modifier is a conversion factor for a physical unit, only the unit type is indicated (length, energy, force, time, etc.). A list of the allowed physical units can be found in appendix C.

2.1 Main input

The input file for DFTB+ (`dftb_in.hsd`) must contain the following property definitions:

Name	Type	Condition	Default	Page
Geometry	p|m		-	10
Hamiltonian	m		-	29

Additionally optional blocks of definitions may be present:

Name	Type	Condition	Default	Page
Driver	m		{}	13
Options	p		{}	79
Analysis	p		{}	82
ExcitedState	p		{}	87
ElectronDynamics	p		{}	91
REKS	p		{}	96
InputVersion	s		None	
ParserOptions	p		{}	100
Parallel	p		{}	100

Geometry Specifies the geometry for the system to be calculated. See p. 10.

Hamiltonian Configures the Hamiltonian and its options. See p. 29.

Driver Specifies a geometry driver for your system. See p. 13.

Options Various global options for the run. See p. 79.

Analysis Post-run analysis and properties options. See p. 82.

ExcitedState Calculations in excited state of the system. See p. 87.

ElectronDynamics Real time electronic dynamics of the system. See p. 91.

REKS Calculations in ensemble DFT for a strongly correlated system. See p. 96.

InputVersion Program version, which the input file was written for.

ParserOptions Various options affecting the parser only. See p. 100.

Parallel Options affecting the MPI-parallel execution. See p. 100.

2.2 Geometry

The geometry can be specified either directly by passing the appropriate list of properties or by using the GenFormat{} method. For molecular input the xyzFormat{} is supported as well. For periodic input the VaspFormat{} supports reading POSCAR and CONTCAR files.

2.2.1 Explicit geometry specification

If the geometry is being specified explicitly, the following properties can be set:

Periodic	l		No
LatticeVectors	9r	Periodic = Yes	-
TypeNames	s+		-
TypesAndCoordinates	(1i3r)+		-

Periodic Specifies if the system is periodic in all 3 dimensions or is to be treated as a cluster. If set to Yes, property LatticeVectors{} must be also specified.

LatticeVectors [*length*] The x, y and z components of the three lattice vectors if the system is periodic.

TypeNames List of strings with the names of the elements, which appear in your geometry.

TypesAndCoordinates [relative|*length*] For every atom the index of its type in the TypeNames list and its coordinates. If for a periodic system (Periodic = Yes) the modifier relative is specified, the coordinates are interpreted in the coordinate system of the lattice vectors.

Example: Geometry of GaAs:

```
Geometry = {
  TypeNames = { "Ga" "As" }
  TypesAndCoordinates [Angstrom] = {
    1  0.000000     0.000000     0.000000
    2  1.356773     1.356773     1.356773
  }
  Periodic = Yes
  LatticeVectors [Angstrom] = {
     2.713546     2.713546     0.
     0.           2.713546     2.713546
     2.713546     0.           2.713546
  }
}
```

2.2.2 GenFormat{}

You can use the generic format to specify the geometry (see appendix D). The geometry specification for GaAs would be the following:

```
Geometry = GenFormat {
  2  S
  Ga  As
  1 1     0.000000     0.000000     0.000000
  2 2     1.356773     1.356773     1.356773
  0.000000     0.000000     0.000000
  2.713546     2.713546     0.
  0.           2.713546     2.713546
  2.713546     0.           2.713546
}
```

It is also possible to include the gen-formatted geometry from a file:

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

2.2.3 xyzFormat{}

You can also use the xyz format to specify molecular geometries. The geometry specification for a caffeine molecule would be the following:

```
Geometry = xyzFormat {
24
caffeine C8H10N4O2
C        1.07317      0.04885     -0.07573
N        2.51365      0.01256     -0.07580
C        3.35199      1.09592     -0.07533
N        4.61898      0.73028     -0.07549
C        4.57907     -0.63144     -0.07531
C        3.30131     -1.10256     -0.07524
C        2.98068     -2.48687     -0.07377
O        1.82530     -2.90038     -0.07577
N        4.11440     -3.30433     -0.06936
C        5.45174     -2.85618     -0.07235
O        6.38934     -3.65965     -0.07232
N        5.66240     -1.47682     -0.07487
C        7.00947     -0.93648     -0.07524
C        3.92063     -4.74093     -0.06158
H        0.73398      1.08786     -0.07503
H        0.71239     -0.45698      0.82335
H        0.71240     -0.45580     -0.97549
H        2.99301      2.11762     -0.07478
H        7.76531     -1.72634     -0.07591
H        7.14864     -0.32182      0.81969
H        7.14802     -0.32076     -0.96953
H        2.86501     -5.02316     -0.05833
H        4.40233     -5.15920      0.82837
H        4.40017     -5.16929     -0.94780
}
```

It is also possible to include the xyz-formatted geometry from a file:

```
Geometry = xyzFormat {
  <<< "geometry.xyz"
}
```

2.2.4 VaspFormat{}

You can also use Vasp's POSCAR or CONTCAR format to specify periodic geometries. Both VASP 4.x and VASP 5.x POSCAR are supported. The geometry specification for a molecular ammonia crystal with VASP 5.x would be the following:

```
Geometry = VaspFormat {
Ammonia
   1.00000000000000
```

```
 5.01336000000000   0.00000000000000   0.00000000000000
 0.00000000000000   5.01336000000000   0.00000000000000
 0.00000000000000   0.00000000000000   5.01336000000000
   H  N
   12 4
Cartesian
 2.19855889440000   1.76390058240000   0.88014548160000
 1.76390058240000   0.88014548160000   2.19855889440000
 0.88014548160000   2.19855889440000   1.76390058240000
 4.84115108400000   1.61941554720000   4.93981400880000
 4.35630903840000   2.49981169680000   3.63248012160000
 3.51957925440000   1.15357413600000   4.08403345680000
 4.08403345680000   3.51957925440000   1.15357413600000
 4.93981400880000   4.84115108400000   1.61941554720000
 3.63248012160000   4.35630903840000   2.49981169680000
 2.49981169680000   3.63248012160000   4.35630903840000
 1.15357413600000   4.08403345680000   3.51957925440000
 1.61941554720000   4.93981400880000   4.84115108400000
 1.37461317840000   1.37461317840000   1.37461317840000
 3.99815460000000   1.99105592400000   4.46364507600000
 4.46364507600000   3.99815460000000   1.99105592400000
 1.99105592400000   4.46364507600000   3.99815460000000
}
```

It is also possible to include the POSCAR/CONTCAR formatted geometry from a file:

```
Geometry = VaspFormat {
  <<< "POSCAR"
}
```

2.3 Driver

The driver is responsible for changing the geometry of the input structure during the calculation. Currently the following methods are available:

{} Static calculation with the input geometry.

GeometryOptimisation{} / GeometryOptimization{} Geometry optimisation of the input geometry. See p. 14.

SteepestDescent{} (**deprecated**) Geometry optimisation by moving atoms along the acting forces. See p. 17.

ConjugateGradient{} (**deprecated**) Geometry optimisation using the conjugate gradient algorithm. See p. 19.

gDIIS{} (**deprecated**) Geometry optimisation using the modified gDIIS method. See p. 19.

LBFGS{} (**deprecated**) Geometry optimisation using the LBFGS algorithm. See p. 20.

FIRE{} (**deprecated**) Geometry optimisation using the FIRE algorithm. See p. 20.

SecondDerivatives{} Calculation of the second derivatives of the energy (the Hessian) and optional first derivatives of other properties (the dipole moment, i.e. Born charges, or the electric polarisability) with respect to atomic coordinates. See p. 21.

VelocityVerlet{} Molecular dynamics with the velocity Verlet algorithm. See p. 22.

Socket{} Hands over control to an external program via a socket interface. See p. 28.

2.3.1 GeometryOptimisation

Optimiser	m		Rational	
MovedAtoms	(ils)+		1:-1	
LatticeOpt	l		No	
FixAngles	l	LatticeOpt = Yes	No	
FixLengths	3l	LatticeOpt = Yes	No No No	
Isotropic	l	LatticeOpt = Yes	No	
Convergence	m			17
MaxSteps	i		200	
OutputPrefix	s		"geo_end"	
AppendGeometries	l		No	

Optimiser / Optimizer Type of the geometry optimiser to use. (See following sections for possible optimiser types.)

MovedAtoms Indices of the atoms which should be moved. The atoms can be specified via index selection expressions, as described in appendix B.7.

LatticeOpt Allow the lattice vectors to change during optimisation. MovedAtoms can be optionally used with lattice optimisation if the atomic coordinates are to be co-optimised with the lattice.[1]

FixAngles If optimising the lattice, allow only the lengths of lattice vectors to vary, not the angles between them. For example if your lattice is orthorhombic, this option will maintain that symmetry during optimisation.

FixLengths If optimising the lattice with FixAngles = Yes, allow only the lengths of the specified lattice vectors to vary.

Isotropic If optimising the lattice, allow only uniform scaling of the unit cell. This option is incompatible with FixAngles.

Convergence Convergence criteria. See section 2.3.1.

MaxSteps Maximum number of steps after which the optimisation should stop (unless already stopped by achieving convergence). Setting this value as -1 runs a huge() number of iterations.

OutputPrefix Prefix of the geometry files containing the final structure.

[1]This is functional but not very efficient at the moment.

AppendGeometries If set to Yes, the geometry file in the XYZ-format will contain all the geometries obtained during the optimisation (instead of containing only the last geometry).

Example for performing a full lattice optimisation using the rational function optimiser.

```
Driver = GeometryOptimisation {
  Optimiser = Rational {}
  LatticeOpt = Yes
}
```

Optimiser: Rational

Provides a rational function based optimiser. The geometry displacement step is determining obtaining the lowest eigenvalue of the augmented Hessian matrix. The eigenvalue equation is solved iteratively using a Davidson solver updating the displacement vector from the last optimisation cycle. The augmented Hessian is build from a guess model hessian, which is updated every cycle from the gradient change and displacement vectors using a BFGS-like scheme. A unit matrix is used as guess for the model hessian.

The scaling behaviour of the computational cost and memory usage for this optimisation step is quadratic in the number of optimised variables. Carefully evaluate the runtime of the optimisation step for (very) large systems together linear-scaling electronic solvers.

DiagLimit	r	1.0e-2

DiagLimit Lower limit of the diagonal Hessian elements in the BFGS-like update step.

```
Driver = GeometryOptimisation {
  Optimiser = Rational { DiagLimit = 1.0e-2 }
}
```

Optimiser: LBFGS

Limited memory BFGS optimiser provides a quasi-Newton optimisation method using limited amount of memory.

The scaling behaviour of the computational cost and memory usage for this optimisation step is linear in the number of optimised variables.

Memory	i	20

Memory Number of past displacements and gradient vectors to keep.

```
Driver = GeometryOptimisation {
  Optimiser = LBFGS { Memory = 20 }
}
```

Optimiser: FIRE

The *fast inertial relaxation engine* of Bitzek *et al.* [13], using the default values from their paper and a Leap-frog integrator. This can be faster than conjugate gradient and competitive with LBFGS in some cases, but you should verify if FIRE is stable and efficient for your system before using it over other optimisers.

The scaling behaviour of the computational cost and memory usage for this optimisation step is linear in the number of optimised variables.

nMin	i	5
aPar	r	0.1
fInc	r	1.1
fDec	r	0.5
fAlpha	r	0.99
StepSize	r	1.0

StepSize [*time*] Step size (δt) for the propagation of the coordinates.

nMin Minimum number of steps before the step size is increased.

aPar Parameter for the update of the velocities.

fInc Factor for the increase of the step size.

fDec Factor for the decrease of the step size.

fAlpha Factor for the update the alpha parameter on reset.

```
Driver = GeometryOptimisation {
  Optimiser = FIRE {
    nMin = 5
    aPar = 0.1
    fInc = 1.1
    fDec = 0.5
    fAlpha = 0.99
    StepSize = 1.0
  }
}
```

Optimiser: SteepestDescent

Steepest descent optimiser, that proposes displacements according to the (scaled) gradient value.

ScalingFactor	r	1.0

ScalingFactor Scaling factor to displace downwards according to the directional derivative.

```
Driver = GeometryOptimization {
  Optimizer = SteepestDescent {
    ScalingFactor = 1.0
  }
}
```

Convergence

Data group to provide the convergence thresholds for the geometry optimisations. To disable a convergence criterium a large value should be provided.

Energy	r	inf
GradElem	r	1e-4
GradNorm	r	inf
DispElem	r	inf
DispNorm	r	inf

Energy [*energy*] Maximum energy change in one geometry step to consider optimisation converged. Disabled by default.

GradElem [*force*] Maximum absolute gradient element in one geometry step to consider optimisation converged Default threshold is 1e-4 Hartree/Bohr.

GradNorm [*force*] Maximum gradient norm in one geometry step to consider optimisation converged Disabled by default.

DispElem [*length*] Maximum absolute displacement element in one geometry step to consider optimisation converged Disabled by default.

DispNorm [*length*] Maximum displacement norm in one geometry step to consider optimisation converged Disabled by default.

2.3.2 SteepestDescent{}

This geometry optimiser is deprecated and will be removed in future versions. Please use the new GeometryOptimisation driver instead.

MovedAtoms	(ils)+		1:-1
MaxForceComponent	r		1e-4
MaxSteps	i		200
StepSize	r		100.0
OutputPrefix	s		"geo_end"
AppendGeometries	l		No
Constraints	(1i3r)*	LatticeOpt = No	{}
LatticeOpt	l	Periodic = Yes	No
FixAngles	l	Periodic = Yes, LatticeOpt = Yes	No
FixLengths	3l	FixAngles = Yes	No No No
Isotropic	l	Periodic = Yes, LatticeOpt = Yes	No
Pressure	r	Periodic = Yes, LatticeOpt = Yes	0.0
MaxAtomStep	r	MovedAtoms $\neq$ None{}	0.2
MaxLatticeStep	r	Periodic = Yes, LatticeOpt = Yes	0.2

MovedAtoms Indices of the atoms which should be moved. The atoms can be specified via index selection expressions, as described in appendix B.7.

MaxForceComponent [*force*] Optimisation is stopped, if the force component with the maximal absolute value goes below this threshold.

MaxSteps Maximum number of steps after which the optimisation should stop (unless already stopped by achieving convergence). Setting this value as -1 runs a huge() number of iterations.

StepSize [*time*] Step size (δt) along the forces. The displacement δx_i along the i^{th} coordinate is given for each atom as $\delta x_i = \frac{f_i}{2m}\delta t^2$, where f_i is the appropriate force component and m is the mass of the atom.

OutputPrefix Prefix of the geometry files containing the final structure.

AppendGeometries If set to Yes, the geometry file in the XYZ-format will contain all the geometries obtained during the optimisation (instead of containing only the last geometry).

Constraints Specifies geometry constraints. For every constraint the serial number of the atom is expected followed by the x, y, z components of a constraint vector. The specified atom is not allowed to move along the constraint vector. If two constraints are defined for the same atom, the atom will only by able to move normal to the plane containing the two constraining vectors.

Example:

```
Constraints = {
  # Atom one can only move along the z-axis
  1  1.0  0.0  0.0
  1  0.0  1.0  0.0
}
```

LatticeOpt Allow the lattice vectors to change during optimisation. MovedAtoms can be optionally used with lattice optimisation if the atomic coordinates are to be co-optimised with the lattice.[2]

FixAngles If optimising the lattice, allow only the lengths of lattice vectors to vary, not the angles between them. For example if your lattice is orthorhombic, this option will maintain that symmetry during optimisation.

FixLengths If optimising the lattice with FixAngles = Yes, allow only the lengths of the specified lattice vectors to vary.

Example:

```
Driver = ConjugateGradient {
  LatticeOpt = Yes
  FixAngles = Yes # Fix angles between lattice vectors
  FixLengths = {Yes Yes No} # Allow only lat. vector 3 to change length
}
```

Isotropic If optimising the lattice, allow only uniform scaling of the unit cell. This option is incompatible with FixAngles.

Pressure [*pressure*] If optimising the lattice, set the external pressure, leading to a Gibbs free energy of the form $G = E + PV - TS$ being printed as well (the included entropy term is only the contribution from the electrons, therefore this is not the full free energy).

MaxAtomStep Sets the maximum possible line search step size for atomic relaxation.

[2]This is functional but not very efficient at the moment.

MaxLatticeStep Sets the maximum possible line search step size for lattice optimisation. For FixAngles or Isotropic calculations this is as a fraction of the lattice vectors or the volume respectively.

2.3.3 ConjugateGradient{}

This geometry optimiser is deprecated and will be removed in future versions. Please use the new GeometryOptimisation driver instead.

MovedAtoms	(ils)+		1:-1
MaxForceComponent	r		1e-4
MaxSteps	i		200
OutputPrefix	s		"geo_end"
AppendGeometries	l		No
Constraints	(1i3r)*		{}
LatticeOpt	l	Periodic = Yes	No
FixAngles	l	Periodic = Yes, LatticeOpt = Yes	No
Isotropic	l	Periodic = Yes, LatticeOpt = Yes	No
Pressure	r	Periodic = Yes	0.0
MaxAtomStep	r	MovedAtoms ≠ None{}	0.2
MaxLatticeStep	r	Periodic = Yes, LatticeOpt = Yes	0.2

See previous subsection for the description of the properties.

2.3.4 gDIIS{}

This geometry optimiser is deprecated and will be removed in future versions. Please use the new GeometryOptimisation driver instead.

Alpha	r		0.1
Generations	i		8
MovedAtoms	(ils)+		1:-1
MaxForceComponent	r		1e-4
MaxSteps	i		200
OutputPrefix	s		"geo_end"
AppendGeometries	l		No
Constraints	(1i3r)*		{}
LatticeOpt	l	Periodic = Yes	No
FixAngles	l	Periodic = Yes, LatticeOpt = Yes	No
Isotropic	l	Periodic = Yes, LatticeOpt = Yes	No
Pressure	r	Periodic = Yes	0.0
MaxLatticeStep	r	Periodic = Yes, LatticeOpt = Yes	0.2

Specific properties for this method are:

Alpha Initial scaling parameter to prevent the iterative space becoming exhausted (this is dynamically adjusted during the run).

Generations Number of generations to consider for the mixing.

See previous subsection for the description of the other properties.[3]

2.3.5 LBFGS{}

This geometry optimiser is deprecated and will be removed in future versions. Please use the new GeometryOptimisation driver instead.

Memory	i		20
LineSearch	l		No
MovedAtoms	(ils)+		1:-1
MaxForceComponent	r		1e-4
MaxSteps	i		200
OutputPrefix	s		"geo_end"
AppendGeometries	l		No
Constraints	(1i3r)*		{}
LatticeOpt	l	Periodic = Yes	No
FixAngles	l	Periodic = Yes, LatticeOpt = Yes	No
Isotropic	l	Periodic = Yes, LatticeOpt = Yes	No
Pressure	r	Periodic = Yes	0.0
MaxLatticeStep	r	LatticeOpt = Yes, LineSearch = Yes or setMaxStep = Yes	0.2
MaxAtomStep	r	LineSearch = Yes or setMaxStep = Yes	0.2

Specific properties for this method are:

Memory Number of last steps which are saved and used to calculate the next step via the lBFGS algorithm. The literature recommends that Memory should between 3 and 20 [70].

LineSearch Should a line search be performed, instead of a quasi-Newton step along the lBFGS direction.

SetMaxStep Limit maximum changes per iteration if the quasi-Newton step along the lBFGS direction is used.

2.3.6 FIRE{}

This geometry optimiser is deprecated and will be removed in future versions. Please use the new GeometryOptimisation driver instead.

The *fast inertial relaxation engine* of Bitzek *et al.* [13], using the default values from their paper and a Leap-frog integrator. This can be faster than conjugate gradient and competitive with LBFGS in some cases, but you should verify if FIRE is stable and efficient for your system before using it over other optimisers.

TimeStep	r	1.0

Specific properties for this method are:

TimeStep [*time*] Controls the maximum step sizexs used in the dynamics integrator.

[3]This approach is distinct from section 2.4.4, but uses a related algorithm based on Ref. [47] and comments from P.R.Briddon.

2.3.7 SecondDerivatives{}

Calculates the second derivatives of the energy, the dipole moment and electric polarisability (currently using a numerical first order finite difference with respect to positions). These can then be used with the MODES code (chapter 5) to calculate the vibrational modes and estimate both their Raman and infrared intensities (see sections 5.1.2 and 5.1.3 for how to do this).

The polarizability derivative can be evaluated either for a static electric field (probably what is usually intended) or at a specific driving frequency (relevant when approaching an electronic resonance).

The hessian matrix is written out for the i, j and k directions of atoms $1 \ldots n$ as

$$\frac{\partial^2 E}{\partial x_{i1}\partial x_{i1}} \; \frac{\partial^2 E}{\partial x_{j1}\partial x_{i1}} \; \frac{\partial^2 E}{\partial x_{k1}\partial x_{i1}} \; \frac{\partial^2 E}{\partial x_{i2}\partial x_{i1}} \; \frac{\partial^2 E}{\partial x_{j2}\partial x_{i1}} \; \frac{\partial^2 E}{\partial x_{k2}\partial x_{i1}} \cdots \frac{\partial^2 E}{\partial x_{kn}\partial x_{kn}}$$

into *hessian.out*

If the dipole moment is available, the Born charges for the n atoms are writen as

$$\begin{array}{ccc}
\frac{\partial^2 E}{\partial \varepsilon_i \partial x_{i1}} & \frac{\partial^2 E}{\partial \varepsilon_j \partial x_{i1}} & \frac{\partial^2 E}{\partial \varepsilon_k \partial x_{i1}} \\
\frac{\partial^2 E}{\partial \varepsilon_i \partial x_{j1}} & \frac{\partial^2 E}{\partial \varepsilon_j \partial x_{j1}} & \frac{\partial^2 E}{\partial \varepsilon_k \partial x_{j1}} \\
\frac{\partial^2 E}{\partial \varepsilon_i \partial x_{k1}} & \frac{\partial^2 E}{\partial \varepsilon_j \partial x_{k1}} & \frac{\partial^2 E}{\partial \varepsilon_k \partial x_{k1}} \\
 & & . \\
 & & . \\
 & & . \\
\frac{\partial^2 E}{\partial \varepsilon_i \partial x_{kn}} & \frac{\partial^2 E}{\partial \varepsilon_j \partial x_{kn}} & \frac{\partial^2 E}{\partial \varepsilon_k \partial x_{kn}}
\end{array}$$

where ε is a cartesian electric field and $\mathbf{x}$ the cartesian directions written into *born.out*

If dipole polarisability is also requested (see section 2.6.4), then the derivatives with respect to atomic coordinates are also calculated. These are stored in the file *bornderiv.out*. Polarisability at only a single static or dynamic frequency should be calculated.

Note: for periodic calculations, all of these derivatives are obtained at the $\mathbf{q} = 0$ point, irrespective of the k-point sampling used.

Important: In order to get accurate results for the second derivatives (and the resulting frequencies) you must set a smaller self-consistent tolerance than the default value in the Hamiltonian{} section. We suggest SCCTolerance = 1e-7 or better. A less accurate tolerance can yield nonphysical vibrational frequencies.

Atoms	i+lm	1:-1
MovedAtoms	i+lm	Value of Atoms
Delta	r	1e-4

Atoms Index of the atoms for which the second derivatives should be computed. The index selection format is described in appendix B.7.

MovedAtoms Optional parameter enabling to select a subset of the atoms which were specified with the Atoms keyword. Only the selected atoms will then be moved in the calculation, yielding a rectangular submatrix ($3\,\mathrm{nMovedAtoms} \times 3\,\mathrm{nAtoms}$) of the full Hessian ($3\,\mathrm{nAtoms} \times 3\,\mathrm{nAtoms}$). This is useful for splitting the Hessian calculation among multiple DFTB+ runs. See also the batch script in tools/misc/set_partial_hessian. If the keyword is not specified, the full Hessian for all atoms specified in Atoms will be calculated. Currently, the atom indices provided in MovedAtoms must build a contiguous range, and also the atom indices listed in Atoms must be contiguous if the MovedAtoms option is used.

Delta [*len*] Optional step size for numerical differentiation of forces to get the second derivatives of the energy with respect to atomic coordinates.

Example:

```
Driver = SecondDerivatives {
  Atoms = 1:12
}
```

2.3.8 VelocityVerlet{}

The code propagates atomic motion using velocity Verlet dynamics with optional thermostats or barostats to control the temperature and/or pressure. Information is printed out during the simulation every MDRestartFrequency steps, and logged in the file md.out (see appendix 3.8).

MovedAtoms	(ils)+		1:-1	
Steps	i		-	
TimeStep	r		-	
KeepStationary	l		Yes	
Thermostat	m		-	23
OutputPrefix	s		"geo_end"	
MDRestartFrequency	i		1	
Velocities	(3r)*		-	
Barostat	m	Periodic = Yes	-	25
Xlbomd	p	XlbomdFast not set		26
XlbomdFast	p	Xlbomd not set		26
Masses	p			28
Plumed	l	No		

MovedAtoms List of atoms to move during the MD. The atoms can be specified via index selection expressions, as described in appendix B.7.

Steps Number of MD steps to perform. In the case of a thermostat using a TemperatureProfile{} the number of steps is calculated from the profile.

KeepStationary Remove translational motion from the system.

TimeStep [*time*] Time interval between two MD steps.

Thermostat Thermostating method for the MD simulation. See p. 23.

OutputPrefix Prefix of the geometry files containing the final structure.

MDRestartFrequency Interval that the current geometry and velocities are written to the XYZ format geometry file and md.out (see section 3.8). In the case of SCC MD runs, the charge restart information is also written at this interval overriding RestartFrequency (see section 2.5).

Velocities [*velocity*] Specified atomic velocities for all the atoms of the given structure (including "velocities" for any stationary atoms, which are silently ignored). This option can be used to restart an MD run, but make sure the geometry is consistent with the specified velocities. The easiest way to do this is to copy both from the same iteration of the XYZ file produced in the previous run (**Note:** the velocities printed in the XYZ files are specified in Å/ps, so this should be set in the input). If restarting an SCC MD run, we **strongly suggest** you use ReadInitialCharges, and in particular read charges for the geometry which you use to restart (iterations at which charges are written to disc are marked in the XYZ file, with the most recent being present in charges.bin or charges.dat depending on the option WriteChargesAsText).

Barostat Berendsen method barostat for the MD simulation. See p. 25.

Xlbomd If present, extended Lagrangian type molecular dynamics is applied to speed up the simulation. For further options within the Xlbomd block see p. 26.

Masses If present, over-ride the atomic masses from the Slater-Koster files. See p. 28

Plumed Whether Plumed should be invoked in order to modify the forces and to drive a meta-dynamics. The parameters of the meta-dynamics must be stored in the file plumed.dat in the current directory. The file must be formatted according to Plumed's own format. (Note: This option requires a DFTB+ binary built with Plumed support.)

Thermostat

None{} No thermostating during the run, only the initial velocities are set according to either a given temperature or velocities, hence an NVE ensemble should be achieved for a reasonable time step.

InitialTemperature	r	-

InitialTemperature [*energy*] Starting velocities for the MD will be created according the Maxwell-Boltzmann distribution at the specified temperature. This is redundant in the case of specified initial velocities.

Andersen{} Andersen thermostat [4] sampling an NVT ensemble.

Note: Andersen thermostating has a reputation for suppressing diffusion and also prevents accumulation of data for dynamical properties.

Temperature	rlm	-
ReselectProbability	r	-
ReselectIndividually	l	-
AdaptFillingTemp	l	No

Temperature [*energy*] Target temperature of the thermostat. It can be either a real value, specifying a constant temperature through the entire run or the TemperatureProfile{} method specifying a changing temperature. (See p. 25.)

ReselectProbability Probability for re-selecting velocities from the Maxwell-Boltzmann distribution.

ReselectIndividually If Yes, each atomic velocity is re-selected individually with the specified probability. Otherwise all velocities are re-selected simultaneously with the specified probability.

AdaptFillingTemp If Yes, the temperature of the electron filling is always set to the current temperature of the thermostat. (The appropriate tag for the temperature of the electron filling is ignored.)

Berendsen{} Berendsen thermostat [12] samples something like an NVT ensemble (but without the correct canonical fluctuations, be aware of the "flying ice cube" problem before using this thermostat [35]).

Temperature	r|m		-
CouplingStrength	r	Timescale not set	-
Timescale	r	CouplingStrength not set	-
AdaptFillingTemp	l		No

Temperature [*energy*] Target temperature of the thermostat. It can be either a real value specifying a constant temperature through the entire run or the TemperatureProfile{} method specifying a changing temperature. (See p. 25.)

CouplingStrength Dimensionless coupling strength for the thermostat (given as $\Delta t/\tau_t$ in the original paper where Δt is the MD time step). Alternatively Timescale[*time*] can be set directly as the characteristic length of time to damp the temperature towards the target temperature. The CouplingStrength and Timescale options are mutually exclusive.

AdaptFillingTemp If Yes, the temperature of the electron filling is always set to the current temperature of the thermostat. (The appropriate tag for the temperature of the electron filling is ignored.)

NoseHoover{} Nosé-Hoover chain thermostat [61] sampling an NVT ensemble, currently with the chain coupled to all of the atoms in the system.

Temperature	r|m	-
CouplingStrength	r	-
ChainLength	i	3
Order	i	3
IntegratorSteps	i	1
Restart	m	
AdaptFillingTemp	l	No

Temperature [*energy*] Target temperature of the thermostat. It can be either a real value, specifying a constant temperature through the entire run or the TemperatureProfile{} method specifying a changing temperature. (See p. 25, but note that profiles are not well tested with this thermostat yet.)

CouplingStrength [*Frequency*] Frequency of oscillation of the thermostating particles (see section 2.5 of Ref. [61]). This is typically related to the highest vibrational mode frequency of the system.

ChainLength Number of particles in the thermostat chain.

Order and **IntegratorSteps** See section 4.3 of Ref. [61]).

Restart Specifies the internal state of the thermostat chain, using three keywords (all three must be present): x{}, v{} and g{} containing the internal chain positions, velocities and forces respectively (these are provided in md.out). See also section 2.3.8.

AdaptFillingTemp If Yes, the temperature of the electron filling is always set to the current temperature of the thermostat. (The appropriate tag for the temperature of the electron filling is ignored.)

TemperatureProfile{} Specifies a temperature profile during molecular dynamics. It takes as argument one or more lines containing the type of the annealing (string), its duration (integer), and the target temperature (real), which should be achieved at the end of the given period. For example:

```
Temperature [Kelvin] = TemperatureProfile {   # Temperatures in K
  constant       1    10.0   # Setting T=10 K for the 0th MD-step
  linear       500   600.0   # Linearly rising T in 500 steps up to T=600 K
  constant     2000   600.0   # Constant T through 2000 steps
  exponential   500    10.0   # Exponential decreasing in 500 steps to T=10 K
}
```

The annealing method can be constant, linear or exponential, with the duration of each stage of the anneal specified in steps of the driver containing the thermostat. The starting temperature for each annealing period is the final target temperature of the previous period, with the last step of each stage being at the target temperature. Since the initial stage in the temperature profile has no previous step, the default starting temperature is set to 0. In order to start the calculation from a finite temperature for the first annealing period, a constant profile temperature stage with a duration of one (or more) step(s) should be specified as the first step (see the example). The temperatures of the stages are specified in atomic units, unless the Temperature keyword carries a modifier, as in the example above.

Barostat

Berendsen barostat [12] samples something like an NPH ensemble for MD (but without the correct fluctuations). Options are provided for either isotropic or cell shape changing pressure control. This can also be used in tandem with a thermostat (p. 23) for the NPT ensemble. If the barostat is used, a partial Gibbs free energy is reported in code output, of the form

$$G = E + PV - TS_{\text{electronic}}.$$

This does not include structural entropy, only any electronic entropy. For barostated constant energy simulations (no thermostat in use), the conserved quantity is the sum of the kinetic and Gibbs partial energies.

Pressure	r		-
Coupling	r	Timescale not set	-
Timescale	r	Coupling not set	-
Isotropic	l		Yes

Pressure [*pressure*] Sets the external target pressure.

Coupling Coupling strength for the barostat (given as $\beta\Delta t/\tau_p$ in the original paper where Δt is the MD time step and β the compressibility, so the coupling is technically dimensioned as reciprocal pressure, but this is usually ignored). Alternatively Timescale[*time*] can be set directly (β/τ_p) as the characteristic length of time to damp the pressure. Typically, β is assumed to be either the experimental value or ~ 1, and Ref. [12] chooses the time scale to be around 10–100 fs. The Coupling and Timescale options are mutually exclusive.

Isotropic Should isotropic scaling of the unit cell be used, or can the cell shape vary? There is a slight inconsistency between the standard forms of these scalings in the literature, which is reproduced here, in brief the isotropic case scales the cell volume by a factor proportional to the differences in the instantaneous and expected pressures (i.e., the cube of the cell vectors), while the anisotropic case changes the cell vectors proportional to the difference instead.

Extended Lagrangian Born-Oppenheimer dynamics

For several systems Born-Oppenheimer molecular dynamics simulations can be significantly sped up by using the extended Lagrangian formalism described in Ref. [7]. The XLBOMD integrator can be used in two different modes:

- Conventional XLBOMD scheme (Xlbomd): The extended Lagrangian is used to predict the input charge distribution for the next time step, instead of taking charges that were converged for the geometry in the previous time step. The predicted starting charges should then require fewer SCC iterations to converge.
- Fast XLBOMD scheme, XlbomdFast (one diagonalisation per time step): The extended Lagrangian is used to predict the population for each time step. This predicted population is then used to build the Hamiltonian, but in contrast to the conventional XLBOMD scheme, there is no self consistent cycle – the forces are calculated immediately after the diagonalisation of the first Hamiltonian. The fast XLBOMD method usually only works for systems without SCC instabilities (e.g. wider gap insulators or molecules without degenerate states). See Ref. [7] for details.

The extended Lagrangian dynamics can be activated by specifying either the Xlbomd or the XlbomdFast option block. Both methods offer the following options:

IntegrationSteps	i	5
PreSteps	i	0

IntegrationSteps Number of time steps used for determining the population for the next time step. Currently, only integration schemes for 5, 6 or 7 steps are implemented.

PreSteps Number of molecular dynamics time steps before the XLBOMD integration becomes activated.

Note: At the step where the XLBOMD integrator becomes active, it is initialised with results of several subsequent converged MD steps, so a further IntegrationSteps + 1 steps will be carried out before the extended Lagrangian dynamics starts to predict the charges and accelerate the calculation.

The conventional Xlbomd block has the following specific options in addition to the common XL-BOMD settings above:

MinSccIterations	i	1
MaxSccIterations	i	200
SccTolerance	r	1e-5

MinSccIterations Minimum number of SCC iterations to perform at each time step during the extended Lagrangian dynamics.

MaxSccIterations Maximum number of SCC iterations to perform at each step in the extended Lagrangian dynamics. If this number of SCC iterations have been reached the forces will be calculated and the MD advances to the next time step. See the note in section 2.4.11 regarding non-convergent k-point sampling.

SccTolerance SCC convergence tolerance during the extended Lagrangian dynamics. Once this tolerance has been achieved the SCC cycle will stop and the forces will be calculated. You can use this parameter to override the SccTolerance parameter in the DFTB block for time steps where the extended Lagrangian integrator is active (This way, you can calculate populations with tight SCC tolerance when initialising the XLBOMD integrator, then use a less strict SCC tolerance once the integrator is active).

The XlbomdFast block has the following specific options in addition to the common XLBOMD settings above:

TransientSteps	i	10
Scale	r	1.0

TransientSteps Enables a smoother transition between Born-Oppenheimer and extended Lagrangian dynamics by carrying out intermediate additional steps with full SCC convergence, during which the converged population and the one predicted by the extended Lagrangian integrator are averaged.

Scale Scaling factor for the predicted charge densities $\in (0, 1]$. The optimal value is system dependent. One should take the highest possible value that still produces stable dynamics (good conservation of energy).

Example for conventional XLBOMD:

```
Xlbomd {
  IntegrationSteps = 6
  MinSccIterations = 2
  MaxSccIterations = 200
  SccTolerance = 1e-6
}
```

Fast (SCC-free) XLBOMD with one diagonalisation per time step:

```
XlbomdFast {
  PreSteps = 5
  TransientSteps = 10
  IntegrationSteps = 5
  Scale = 0.5
}
```

Points to be aware of:

- The extended Lagrangian (especially in the fast mode) needs special force evaluation giving more accurate forces for non-convergent charges. Therefore you must set the ForceEvaluation option to 'Dynamics' (for simulations with finite electronic temperature) or to 'DynamicsT0' (for simulations at 0 K electronic temperature) in the DFTB block (see p. 79).
- The extended Lagrangian implementation only works for the (N,V,E) ensemble so far, so neither thermostats nor barostats are allowed.
- The extended Lagrangian implementation currently cannot be used for spin-polarised or spin-orbit systems, or when electron filling methods other than Fermi{} filling (see p. 48) are used.

Masses

Provides values of atomic masses for specified atoms, ranges of atoms or chemical species. This is useful for example to set isotopes for specific atoms in the system.

Mass	p

Any atoms not given specified masses will use the default values from the appropriate homonuclear Slater-Koster file. An example is given below

```
Masses {
  Mass {
    Atoms = 1:2
    MassPerAtom [amu] = 2.0
  }
}
```

where Atoms specifies the atom or atoms which each have a mass of MassPerAtom assigned.

2.3.9 Socket{}

The code tries to connect to a socket interface to receive control instructions from an external driver code.

File	s	Host not set	-
Prefix	s	Host not set	"/tmp/ipi_" for Protocol = i-PI{}
Host	s	File not set	-
Port	i	File is set	-
Verbosity	i		0
Protocol	m		i-PI{}
MaxSteps	i		200

File Name of UNIX style file socket to connect to.

Prefix Prefix to the file name, in the case of i-PI dialogue, the defaults to the path and file start that i_PI expects.

Host Name or ip address of internet host to connect to ("localhost" also possible).

Port Port of host to connect to.

Verbosity Level of port traffic to document.

Protocol Choice of message protocol over the socket connection (only communication with i-PI [17] is currently supported).

MaxSteps Number of geometry steps before termination of the DFTB+ instance. Setting this value as -1 runs a huge() number of iterations.

Examples

First an ip address connection:

```
Driver = Socket {
  Host = localhost
  Port = 21012 # port number
  Verbosity = 0 # minimal verbosity
  Protocol = i-PI {} # i-PI interface
  MaxSteps = -1 # Run indefinitely
}
```

Then a UNIX socket via the /tmp file system

```
Driver = Socket {
  File = "dftb" # The protocol defines a default path in this case
  Protocol = i-PI {} # i-PI interface
  MaxSteps = 1000 # Terminate this instance after 1000 steps
}
```

Please note that this driver changes the default behaviour of some options to remove (usually unneeded) file writing: WriteDetailedOut = No and WriteBandOut = No.

2.4 Hamiltonian

Currently, DFTB and xTB (section 2.4.2) Hamiltonians are implemented.

2.4.1 Density Functional based Tight Binding

To select the DFTB Hamiltonian you must set Hamiltonian = DFTB{}. The DFTB{} method may contain the following properties:

SCC	l		No	
SCCTolerance	r	SCC = Yes	1e-5	
MaxSCCIterations	i	SCC = Yes	100	
ConvergentSccOnly	l	SCC = Yes	Yes	
EwaldParameter	r	Periodic = Yes SCC = Yes	0.0	
EwaldTolerance	r	Periodic = Yes SCC = Yes	1e-9	
ShellResolvedSCC	l	SCC = Yes	No	
ElectronicConstraints	p		{}	37
Mixer	m	SCC = Yes	Broyden{}	38
MaxAngularMomentum	p		-	
Charge	r		0.0	
SpinPolarisation	m	SCC = Yes	{}	40
SpinConstants	p	SpinPolarisation ≠ {}	-	43
ShellResolvedSpin	l	ShellResolvedSCC = No	No	
SpinOrbit	m	SpinPolarisation ≠ Colinear{}	{}	44
Solver	m		RelativelyRobust{}	44
Filling	m		Fermi{}	47
NonAufbau{}	m		-	49
SlaterKosterFiles	plm		-	50
OldSKInterpolation	l		No	
PolynomialRepulsive	plm		{}	
Chimes	p			
KPointsAndWeights	(4r)+lm	Periodic = Yes	-	51
OrbitalPotential	m	SpinPolarisation ≠ {}	{}	54
ReadInitialCharges	l	SCC = Yes	No	
InitialCharges	p	SCC = Yes	{}	
ElectricField	p		{}	54
AtomSitePotential	p		{}	56
Dispersion	m		{}	57
HCorrection	m	SCC = Yes	None {}	75
HalogenXCorr	l	ThirdOrder(Full) = Yes, DftD3	No	75
ThirdOrder	l	SCC = Yes	No	
ThirdOrderFull	l	SCC = Yes	No	66
RangeSeparated	p		None	77
HubbardDerivs	p	ThirdOrder(Full) = Yes	-	
OnSiteCorrection	p	SCC = Yes	-	78
Solvation	m		-	67
Differentiation	m		FiniteDiff	79
ForceEvaluation	s		"Legacy"	
CustomisedHubbards	p	SCC = Yes		
CustomisedOccupations	p	SCC = Yes		
TruncateSKRange	p			

SCC If set to Yes, a self consistent charge (SCC) calculation is made.

SCCTolerance Stopping criteria for the SCC. Specifies the tolerance for the maximum difference in any charge between two SCC cycles.

MaxSCCIterations Maximal number of SCC cycles to reach convergence. If convergence is not reached after the specified number of steps, the program stops. However in cases where the calculation is not for a static structure (so Driver $\neq$ {}), this behaviour can be overridden using ConvergentSccOnly. **Note:** For calculations using k-points, the default number of SCC iterations will default to 1 in cases where a band structure is being plotted (see KLines{} in section 2.4.11).

ConvergentSccOnly If true, requires that the SCC cycle converges before proceeding to subsequent stages of the calculation (forces, analysis, etc.).

EwaldParameter Sets the dimensionless parameter α in the Ewald electrostatic summation for periodic calculations. This controls the fraction of the Ewald summation occurring in real and reciprocal space. Setting it to zero or negative activates an automatic determination of this parameter (default and recommended behaviour). Setting it positive forces the code to use the supplied value. This is useful for very asymmetrical unit cells (typically a slab or nanowire with a huge surrounding vacuum region) since the memory demand of DFTB+ can increase dramatically in these cases (due to storage of a long range real space neighbour list). To determine a suitable value of α for such a cell, you should initially reduce the vacuum region and run a test calculation, looking for the value of the automatically chosen Ewald parameter in the standard output. This is then a suitable choice for the original cell with the large vacuum region.

EwaldTolerance Sets the tolerance for Ewald summation in periodic systems.

ShellResolvedSCC If set to Yes, all distinct Hubbard U values for the different atomic angular momenta shells are used, when calculating the SCC contributions. Otherwise, the value supplied for the s-shell is used for all angular momenta. Please note, that the old standard DFTB code was *not* orbitally resolved, so that only the Hubbard U for the s-shell was used. Please check the documentation of the SK-files you intend to use as to whether they are compatible with an orbitally resolved SCC calculation (many of the biological files do not use orbitally resolved charges), before you switch this option to Yes. Even if the Hubbard U values for different shells are the same in the SK-files, this flag would still effect your results, since when it is set to Yes, any charge transfer between atomic shells will change the energy of the system compared to when it is set to set to No.

Mixer Mixer type for mixing the charges in an SCC calculation. See p. 38.

MaxAngularMomentum Specifies the highest angular momentum for each atom type. All orbitals up to that angular momentum will be included in the calculation. Several main-block elements require d-orbitals, check the documentation of the SK-files you are using to determine if this is necessary. Possible values for the angular momenta are s, p, d, f.

Example:

```
MaxAngularMomentum = {
  Ga = "p"    # You can omit the quotes around the
  As = "p"    # orbital name, if you want.
}
```

By using the SelectedShells method it is also possible to combine shells from different Slater-Koster files together to treat atoms containing multiple shells with the same angular momentum. The shells to be picked from a particular atom type should be listed without any separating characters. The list of shells of different atom types should be separated by white spaces.

Example:

```
# Defining sps* basis for Si and C by combining sp and s shells from
# Si and Si2, and C and C2, resp.
MaxAngularMomentum = {
  Si = SelectedShells { "sp" "s" }    # Si atom with sps* basis
  C = SelectedShells { "sp" "s" }     # C atom with sps* basis
}

# Note, that you have to modify the Slater-Koster file definition accordingly
SlaterKosterFiles = {
  Si-Si = "Si-Si.skf" "Si-Si2.skf" "Si2-Si.skf" "Si2-Si2.skf"
  Si-C = "Si-C.skf" "Si-C2.skf" "Si2-C.skf" "Si2-C2.skf"
  C-Si = "C-Si.skf" "C-Si2.skf" "C2-Si.skf" "C2-Si2.skf"
  C-C = "C-C.skf" "C-C2.skf" "C2-C.skf" "C2-C2.skf"
}
```

If for a given atomic type you pick orbitals from more than one species, you must specify an appropriate combinations of file names for the Slater-Koster tables in SlaterKosterFiles{}. For every atom type combination $n_{\mathrm{SK1}} \times n_{\mathrm{Sk2}}$ Slater-Koster files must be defined, where n_{SK1} and n_{SK2} are the number species combined to build up the shells of the two interacting atoms. The file names must be ordered with respect to the interacting species, so that the name for the second interacting species is changed first. Above you see an example, where an extended basis with an s^*-orbital was generated by introducing the new species "Si2" and "C2", containing the appropriate s^*-orbital for Si and C, resp., as only orbitals.

In the case of multiple Slater-Koster files for a certain interaction, the repulsive data is read from the first specified file (e.g. Si-Si.skf, Si-C.skf, C-Si.skf and C-C.skf in the example above). The repulsive interactions in the other files are ignored. The mass for a certain species is read from the first SK-file for its homo-nuclear interaction.

Non-minimal basis Slater-Koster data may be directly defined in the SK-files in future.

Charge Total charge of the system in units of the electron charge. Negative values mean an excess of electrons. If the keyword FixedFermiLevel is present (see section 2.4.8), then value specified here will be ignored.

SpinPolarisation / SpinPolarization Specifies if and how the system is spin polarised. See p. 40.

SpinConstants Specifies the atom type specific constants needed for the spin polarised calculations, in units of Hartree. See p. 43.

SpinOrbit Specifies if the system includes Russel-Saunders coupling. See p. 44

Solver Specifies which solver (eigensolver, Green's function, etc.) to use with the hamiltonian. See p. 44.

Filling Method for occupying the one electron levels with electrons. See p. 47.

SlaterKosterFiles Name of the Slater-Koster files for every atom type pair combination. See 50.

OldSKInterpolation If set to Yes (strongly discouraged), the look-up tables for the overlap and non-scc Hamiltonian contribution are interpolated with the same algorithm as in the *old* DFTB code. Please note, that the new method uses a smoother function, is more systematic, and yields better derivatives than the old one. This option is present only for compatibility purposes, and may be removed in the future.

PolynomialRepulsive Specifies for each interaction, if the polynomial repulsive function should be used. for every pairwise combination of atoms it should contain a logical value, where Yes stands for the use of a polynomial repulsive function and No for a spline. If a specific pair of species is not specified, the default value No is used.

Example:

```
# Use the polynomial repulsive function for Ga-Ga and As-As interactions
# in GaAs
PolynomialRepulsive = {
  Ga-Ga = Yes
  Ga-As = No
  # As-Ga unspecifed, therefore per default set to No
  As-As = Yes
}
```

If you want to apply the same setting for all species pairs, you can specify the appropriate logical value as argument of the SetForAll keyword:

```
# Using polynomial repulsive functions for all interactions in GaAs
PolynomialRepulsive = SetForAll { Yes }
```

Chimes If specified, a ChIMES force field correction will be applied on top of the repulsive potentials found in the SK-files, also known as DFTB/ChIMES model [28]. The Chimes block contains ParameterFile as only keyword, where the file with the ChIMES parameters must be specified. The file is additionally searched for in the directory specified with the DFTB-PLUS_PARAM_DIR environment variable.

Example:

```
    Chimes {
        ParameterFile = "chimes/CNOH.txt"
    }
```

Note: This option requires a DFTB+ binary built with ChIMES support.

KPointsAndWeights [relative|absolute] Contains the special k-points to be used for the Brillouin-zone integration. See p. 51. For automatically generated k-point grids the modifier should not be set.

OrbitalPotential Specifies which (if any) orbitally dependant contributions should be added to the DFTB energy and band structure. See p. 54.

ReadInitialCharges If set to Yes the first Hamiltonian is constructed by using the charge information read from the file charges.bin or charges.dat (depending on the option WriteChargesAsText, see section2.5).

InitialCharges Specifies initial charges, either for all atoms or for only selected ones. In order to specify it for all atoms, use the keyword AllAtomCharges and supply the charge for every atom as a list of real values:

```
InitialCharges = {
  AllAtomCharges = { # Specifies charge for each atom in an H2O molecule
    -0.88081627    # charge for atom 1 (O)
     0.44040813    # charge for atom 2 (H1)
     0.44040813    # charge for atom 3 (H2)
  }
}
```

Alternatively you can specify charges individually for atoms or species using the AtomCharge keyword. For every AtomCharge declaration you must provide a charge and the list of atoms, which should be initialised to that charge. (The atoms can be specified via index selection expressions, as described in appendix B.7.):

```
InitialCharges = { # Specifying charge for various species
  AtomCharge = {
    Atoms = H
    ChargePerAtom = 0.44040813
  }
  AtomCharge {
    Atoms = O
    ChargePerAtom = -0.88081627
  }
}
```

Charge on atoms not appearing in any AtomCharge specification is set to be zero.

ElectricField Specifies an external electric field, arising currently from either an applied field or a distribution of electrostatic charges. See p. 54.

AtomSitePotential Specifies an external potential at an atom. See p. 56.

Dispersion Specifies which kind of dispersion correction to apply. See p. 57.

OnSiteCorrection Used to include the on-site matrix elements of Domínguez *et al.* [20]. See p. 78.

Differentiation Specifies how to calculate finite difference derivatives in the force routines. See p. 79.

ForceEvaluation Decides which expressions are used to calculate the ground state electronic forces. See p. 79. **Note:** all methods give the same final forces when the charges are well converged. For non-converged charges however the resulting forces can differ considerably between methods.

CustomisedHubbards / CustomizedHubbards Enables overriding of the Hubbard U values for given species. If the option ShellResolvedScc has been set to Yes, you need to specify one Hubbard U value for each atomic shell in the basis of the given atom type, otherwise only one atomic value is required. For all species not specified in this block, the value(s) found in their respective Slater-Koster files will be used.

Warning: This option is for experts only! Overriding values stored in the SK-files may result in **inconsistent results**. Make sure you understand the consequences when using this option.

Example:

```
CustomisedHubbards {
  Si = 0.32 0.24
}
```

CustomisedOccupations / CustomizedOccupations Enables overriding the reference neutral atom electronic occupations of the shells. Note that the atom remains neutral since a corresponding ionic counter charge is implicitly added to its core. This option can be used, for example, to simulate effective doping by setting a slightly larger or smaller number of electrons on certain atoms.

Example:

```
CustomisedOccupations{
  ReferenceOccupation{
    Atoms={1:30}
    p=2.01
  }
  ReferenceOccupation{
    Atoms={31:60}
    p=1.99
  }
}
```

The example above sets a filling population of +0.01e or -0.01e in the p shell of the corresponding atom indices. When the states are filled up, the electron excess or depletion results in a shift of the Fermi level in the bands.

Warning: This option is for experts only! Overriding values stored in the SK-files may result in **inconsistent results**. Please look at the transport section of the dftb+ recipes to see an example of the correct use of this option.

TruncateSKRange Enables overriding of the number of elements to be read from the Slater-Koster parameters, shortening the interaction range of atoms.

Warning: This option is for experts only! Overriding values stored in the SK-files may result in **inconsistent results**. Make sure you understand the consequences when using this option.

SKMaxDistance	r	–
HardCutOff	l	No

SKMaxDistance [*length*] Length at which to cut the overlap and non-SCC interactions for all atoms in the system. If this length is longer than the distances in the Slater-Koster files it will have no effect.

HardCutOff The Slater-Koster interpolation DFTB+ produces will smoothly tail off to zero at a short distance beyond the table, which is controlled by OldSKInterpolation. If HardCutOff is set to Yes, then the distance set by SKMaxDistance includes this tail, i.e., no interactions occur beyond that distance. In the case of No this zeroing tail extends slightly beyond the HardCutOff distance.

Example:

```
TruncateSKRange = {
  SKMaxDistance [AA] = 4.0
  HardCutOff = Yes
}
```

2.4.2 Extended Tight Binding Hamiltonian

The extended tight binding (xTB) Hamiltonian[9] is available if DFTB+ is built with support for a communication bridge to the tblite library.

Method	s			
ParameterFile	s			
SCCTolerance	r		1e-5	
MaxSCCIterations	i		100	
MaxAngularMomentum	p		-	
Charge	r		0.0	
SpinPolarisation	m		{}	40
SpinConstants	p	SpinPolarisation $\neq$ {}	-	43
Solver	m		RelativelyRobust{}	44
Filling	m		Fermi{}	47
KPointsAndWeights	(4r)+|m	Periodic = Yes	-	51

Method Request library to initialise the xTB parametrisation for the given method. Doing so will hand over the energy, potential and force evaluation to the tblite library. Supported methods are GFN1-xTB[31], IPEA1-xTB[8], and GFN2-xTB[10]. If present ParameterFile must not be provided.

ParameterFile Request library to initialise the xTB parametrisation from a parameter file. Doing so will hand over the energy, potential and force evaluation to the tblite library. The parameter format is documented in https://tblite.readthedocs.io/en/latest/spec/parameter.html and can be generated using the tblite command line driver. If present Method must not be provided.

This file is additionally searched for in the directory specified with the DFTBPLUS_PARAM_DIR environment variable.

SCCTolerance Stopping criteria for the SCC. Specifies the tolerance for the maximum difference in any charge between two SCC cycles.

MaxSCCIterations Maximal number of SCC cycles to reach convergence. If convergence is not reached after the specified number of steps, the program stops. However in cases where the calculation is not for a static structure (so Driver $\neq$ {}), this behaviour can be overridden using ConvergentSccOnly.

Charge Total charge of the system in units of the electron charge. Negative values mean an excess of electrons. If the keyword FixedFermiLevel is present (see section 2.4.8), then value specified here will be ignored.

SpinPolarisation / SpinPolarization Specifies if and how the system is spin polarised. See p. 40.

SpinConstants Specifies the atom type specific constants needed for the spin polarised calculations, in units of Hartree. See p. 43.

Solver Specifies which solver (eigensolver, Green's function, etc.) to use with the hamiltonian. See p. 44.

Filling Method for occupying the one electron levels with electrons. See p. 47.

KPointsAndWeights [relative|absolute] Contains the special k-points to be used for the Brillouin-zone integration. See p. 51. For automatically generated k-point grids the modifier should not be set.

```
Hamiltonian = xTB {
  Method = "GFN1-xTB"
}
```

2.4.3 ElectronicConstraints

Allows to impose constraints on the electronic ground state by finding the ground state of the system in a (carefully chosen) external potential [38].

Regions	p	{}
Optimiser	m	FIRE
ConstrTolerance	r	1e-8
MaxConstrIterations	i	100
ConvergentConstrOnly	l	Yes

Regions Specifies region (e.g. of atoms) targeted by electronic constraints.

Atoms	p	{}

Atoms Defines a region of atoms (the specification of several blocks is possible).

Domain	(ils)+
Population	r

Domain Indices of the atoms building a constrained region. The atoms can be specified via index selection expressions, as described in appendix B.7.

Population Sets constraint on the Mulliken population of all atoms included in the region.

Optimiser The optimiser is responsible for propagating the micro-iterations, while searching for an optimal auxiliary potential that enforces the electronic constraints set in the Regions{} block. Currently the following methods are available:

SteepestDescent{} Propagate micro-iterations using the steepest descent algorithm. See p. 16.

FIRE{} Propagate micro-iterations using the FIRE algorithm. See p. 15.

Rational{} Propagate micro-iterations using a rational function based optimiser. See p. 15.

LBFGS{} Propagate micro-iterations using the LBFGS algorithm. See p. 15.

Please note, that the SteepestDescent{} and FIRE{} optimiser proved to be most stable, when propagating the micro-iterations. However, your experience may vary and manual testing based on your particular system is advised.

ConstrTolerance Stopping criteria for the micro-iterations. Specifies the tolerance for the maximum difference in any condition of Regions{} between two micro-iteration cycles.

MaxConstrIterations Maximal number of micro-iterations to reach convergence. If convergence is not reached after the specified number of steps, the program stops.

ConvergentConstrOnly If true, requires that the micro-iterations converge before proceeding to subsequent stages of the calculation (next SCC step and eventually forces, analysis, etc.).

Example:

```
ElectronicConstraints {
  Regions {
    Atoms {
      Domain = 1
      Population = 1.5
    }
  }
  Optimiser = FIRE {}
  ConstrTolerance = 1.0e-08
  MaxConstrIterations = 30
  ConvergentConstrOnly = Yes
}
```

2.4.4 Mixer

DFTB+ currently offers the charge mixing methods Broyden{}, Anderson{}, DIIS{} (DIIS accelerated simple mixer) and Simple{} (simple mixer).

Broyden{}

Minimises the error function

$$E = \omega_0^2 \left| G^{(m+1)} - G^{(m)} \right| + \sum_{n=1}^{m} \omega_n^2 \left| \frac{n^{(n+1)} - n^{(n)}}{|F^{(n+1)} - F^{(n)}|} + G^{(m+1)} \frac{F^{(n+1)} - F^{(n)}}{|F^{(n+1)} - F^{(n)}|} \right|^2 ,$$

where $G^{(m)}$ is the inverse Jacobian, $n^{(m)}$ and $F^{(m)}$ are the charge and charge difference vector in iteration m. The weights are given by ω_0 and ω_m, respectively. The latter is calculated as

$$\omega_m = \frac{c}{\sqrt{F^{(m)} \cdot F^{(m)}}}, \tag{2.1}$$

c being a constant coefficient [41].

The Broyden{} method can be configured using following properties:

MixingParameter	r	0.2
InverseJacobiWeight	r	0.01
MinimalWeight	r	1.0
MaximalWeight	r	1e5
WeightFactor	r	1e-2

MixingParameter Mixing parameter.

InverseJacobiWeight Weight for the difference of the inverse Jacobians (ω_0).

MinimalWeight Minimal allowed value for the weighting factors ω_m.

MaximalWeight Maximal allowed value for ω_m.

WeightFactor Weighting factor c for the calculation of the weighting factors ω_m in (2.1).

Note: As the Broyden-mixer stores a copy of the mixed quantity for each SCC iteration at a given geometry, you may consider to choose a different mixer with lower memory requirements, if your system needs density matrix mixing (e.g. DFTB+U), is large and needs a high number of SCC-iterations (MaxSCCIteration).

Anderson{}

Modified Anderson mixer [24].

MixingParameter	r	0.05
Generations	i	4
InitMixingParameter	r	0.01
DynMixingParameters	(2r)*	{}
DiagonalRescaling	r	0.01

MixingParameter Mixing parameter.

Generations Number of generations to consider for the mixing. Setting it too high can lead to linearly dependent sets of equation.

InitMixingParameter Simple mixing parameter used until the number of iterations is greater or equal to the number of generations.

DynMixingParameters Allows specification of different mixing parameters for different levels of convergence during the calculation. These are given as a list of tolerances and the mixing factor to be used below this level of convergence. If the loosest specified tolerance is reached, the appropriate mixing parameter supersedes that specified in MixingParameter.

DiagonalRescaling Used to increase the diagonal elements in the system of equations solved by the mixer. This can help to prevent linear dependencies occurring during the mixing process. Setting it to too large a value can prevent convergence. (This factor is defined in a slightly different way from Ref. [24]. See the source code for more details.)

Example:

```
Mixer = Anderson {
  MixingParameter = 0.05
  Generations = 4
  # Now the over-ride the (previously hidden) default old settings
  InitMixingParameter = 0.01
  DynMixingParameters = {
    1.0e-2  0.1 # use 0.1 as mixing if more converged that 1.0e-2
    1.0e-3  0.3 # again, but 1.0e-3
    1.0e-4  0.5 # and the same
  }
  DiagonalRescaling = 0.01
}
```

DIIS{}

Direct inversion of the iterative space is a general method to acceleration iterative sequences. The current implementation accelerates the simple mix process.

InitMixingParameter	r	0.2
Generations	i	6
UseFromStart	l	Yes

MixingParameter Mixing parameter.

Generations Number of generations to consider for the mixing.

UseFromStart Specifies if DIIS mixing should be done right from the start, or only after the number of SCC-cycles is greater or equal to the number of generations.

Simple{}

Constructs a linear combination of the current input and output charges as $(1-x)q_{\text{in}} + xq_{\text{out}}$.

MixingParameter	r	0.05

MixingParameter Coefficient used in the linear combination.

2.4.5 SpinPolarisation

In an SCC calculation, the code currently supports three different choices for spin polarisation; non-SCC calculations are not spin polarised.

No spin polarisation ({})

No spin polarisation contributions to the energy or band-structure.

Colinear{}

Colinear spin polarisation in the z direction. The following options can be specified:

UnpairedElectrons	r	0
RelaxTotalSpin	l	No
InitialSpins	p	{}

UnpairedElectrons Number of unpaired electrons. This is kept constant during the run, unless the RelaxTotalSpin keywords says otherwise.

RelaxTotalSpin If set to Yes, a common Fermi-level is used for both spin channels, so that the total spin polarisation can change during run. In this case, the spin polarisation specified using the UnpairedElectrons keyword is only applied at initialisation. If set to No (default), the initial spin polarisation is kept constant during the entire run.

InitialSpins Optional initialisation for spin patterns. If this keyword is present, the default code behaviour is that the initial input charge distribution has a magnetisation of 0. Otherwise if it is not present, the initial input charge distribution has a magnetisation matching the number of UnpairedElectrons keyword.

The initial spin distribution for the input charges can be set by specifying the spin polarisation of atoms. For atoms without an explicit specification, a spin polarisation of zero is assumed. The InitialSpins property block must contain either the AllAtomSpins keyword with a list of spin polarisation values for every atom, or one or more AtomSpin blocks giving the spin for a specific group of atoms using the following keywords:

Atoms	(ils)+	-
SpinPerAtom	r	-

Atoms Atoms to specify an initial spin value. The atoms can be specified via indices, index ranges and/or species. (The atoms can be specified via index selection expressions, as described in appendix B.7.)

SpinPerAtom Initial spin polarisation for each atom in this InitialSpins block.

For atoms not appearing in any of the SpinPerAtom block, an initial spin polarisation of 0 is set.

Example (individual spin setting):

```
SpinPolarisation = Colinear {
  UnpairedElectrons = 0.0
  InitialSpins = {
    AtomSpin = {
      Atoms = 1:2
      SpinPerAtom = -1.0
    }
    AtomSpin = {
      Atoms = 3:4
      SpinPerAtom = +1.0
    }
  }
}
```

Example (setting all spins together):

```
SpinPolarisation = Colinear {
  UnpairedElectrons = 0.0
  InitialSpins = {
    AllAtomSpins = { -1.0 -1.0 1.0 1.0 } # Atoms 1,2: -1.0, atoms 3,4: 1.0
  }
}
```

NonColinear{}

Non-collinear spin polarisation with arbitrary spin polarisation vector on every atom. The only option allowed is to set the initial spin distribution:

InitialSpins	p	{}

InitialSpins Initialisation of the x, y and z components of the initial spins for atoms. The default code behaviour is an initial spin polarisation of (0 0 0) for every atom.

The initial spin distribution can be set by specifying the spin polarisation vector on all atoms using the AllAtomSpins keyword or by using one or more AtomSpin blocks with the following options:

Atoms	(i|s)+	-
SpinPerAtom	(3r)+	-

Atoms Atoms to specify an initial spin vector. The atoms can be specified via index selection expressions, as described in appendix B.7.

SpinPerAtom Initial spin polarisation for each atom in this InitialSpins block.

For atoms not appearing in any of the SpinPerAtom block, the vector (0 0 0) is set.

Please note, that in contrast to the collinear case, in the non-collinear case you must specify the spin vector (3 components!) for the atoms.

Example:

```
SpinPolarisation = NonColinear {
  InitialSpins = {
    # Setting the spin for all atoms in the system
    AllAtomSpins = {
      0.408 -0.408  0.816
      0.408 -0.408  0.816
     -0.408  0.408 -0.816
     -0.408  0.408 -0.816
    }
  }
}
```

Example:

```
SpinPolarisation = NonColinear {
  InitialSpins = {
    AtomSpin = {
      Atoms = 1:2
      SpinPerAtom = 0.408 -0.408 0.816
    }
    AtomSpin = {
      Atoms = 3:4
      SpinPerAtom = -0.408  0.408 -0.816
    }
  }
}
```

SpinConstants

This environment suplies the atomic constants required for either spin polarised calculations or when evaluating properties which depend on spin interactions (triplet excitations for example). In these cases, for each atomic species in the calculation the spin coupling constants for that atom must be specified.

When ShellResolvedSCC = No, an extra keyword in this block controls whether the spin constants are resolved by shell or are identical for all shells: ShellResolvedSpin, defaulting to the same value as ShellResolvedSCC.

When shell resolved spin constants are specified, they must be ordered with respect to the pairs of shells they couple, such that the index for the second shell increases faster. For an *spd*-basis, that gives the following ordering:

$$w_{ss}, w_{sp}, w_{sd}, \ldots, w_{ps}, w_{pp}, w_{pd}, \ldots, w_{ds}, w_{dp}, w_{dd}, \ldots$$

Example (GGA parameters for H_2O):

```
SpinConstants = {
  O = {
    # Wss  Wsp    Wps    Wpp
    -0.035 -0.030 -0.030 -0.028
  }
  H = {
    # Wss
    -0.072
  }
}
```

Several standard values of atomic spin constants are given in appendix E. Constants calculated with the same density functional as the SK-files should be used. This input block may be moved to the SK-data definition files in the future.

When using the SelectedShells method for the keyword MaxAngularMomentum, the spin constants are listed as an array of values running over SK1SK2... in the same order as listed for SlaterKoster-Files.

```
SpinConstants = { # not real values, only an example
  Si = {
    # Wss  Wsp    Wss*
    -0.035 -0.030 -0.01
    # Wps  Wpp    Wps*
    -0.030 -0.037 -0.02
    # Ws*s Ws*p   Ws*s*
    -0.01  -0.02  -0.01
  }
```

For cases where ShellResolvedSpin = No, the spin constant for the highest occupied orbital of each atom should be supplied: Example (GGA parameters for H_2O):

```
SpinConstants = {
  O = {
    #Wpp
    -0.028
  }
  H = {
    # Wss
    -0.072
  }
}
```

2.4.6 SpinOrbit

If present, specifies that $L \cdot S$ coupling should be included for a calculation. Currently spin unpolarised and non-collinear spin polarisation are supported, but not collinear spin polarisation. For every atomic species present in the calculation the spin-orbit coupling constants, ξ, for that atom must be specified for all shells present. The constants must be ordered with respect to the list of shells for the given atom.

In the case that the spin-orbit constant for an s orbital has been set to be a non-zero value the code prints a warning. For periodic systems, use of this keyword automatically prevents the folding by inversion normally used in SupercellFolding{}, but manually specified KPointsAndWeights should *not* be reduced by inversion.

Example (GaAs):

```
SpinOrbit = {
  Ga [eV] = {0.0 0.12 0.0} # s p d shells
  As [eV] = {0.0 0.32703} # s p shells
}
```

The additional option in this block, Dual, sets whether to use a block population for the local spin matrices consistent with the dual populations of Han *et al.* [32] or the conventional on-site part of the single particle density matrix. The default value of this option is Yes, also giving extra information regarding atomic orbital moments in the detailed output.

2.4.7 Solver

Currently the following LAPACK 3.0 [5] eigensolver methods are always available:

- QR{}
  (QR decomposition based solver)
- DivideAndConquer{}
  (this requires about twice the memory of the other solvers)
- RelativelyRobust{}
  (using the subspace form but calculating all states)
- MAGMA{}
  (Only available for DFTB+ binaries compiled with MAGMA [90, 91, 21] GPU support. **WARNING:** this is currently an experimental feature, so should be used with care.)

None of these solvers need any parameters or properties to be specified.

Example:

```
Solver = DivideAndConquer {}
```

For ScaLAPACK enabled compilation, all three solvers are also available for MPI parallel use.

If DFTB+ is compiled with the ELSI library also included [94], the additional ELPA, OMM, PEXSI and NTPoly solvers also become available.

Note: The ELSI-solvers are not tested with multiple OpenMP-threads. Therefore, DFTB+ will stop with an error, if an ELSI-solver has been selected and the maximal number of allowed threads is greater than one. (You can control the number of allowed OpenMP-threads via the OMP_NUM_THREADS environment variable.)

ELPA

The ELPA solver provides an efficient and scalable diagonaliser, which is also able to utilise GPUs (provided the ELSI library was compiled with GPU support). It accepts following options

Autotune	l	No
Gpu	l	No
Mode	i	2
Sparse	l	No

Autotune Whether ELPAs autotune capability should be used to optimise performance (default: No)

Gpu Whether ELPA should use available GPUs (default: No). Note, that you can only enable this option, if DFTB+ was built with GPU support.

Mode ELPA operation mode (possible values: 1, 2, default: 2). Mode 2 is supposed to be more efficient for large problems.

Sparse Whether the sparse matrix interface of ELPA should be used or the dense one (default: dense). The sparse interface does not reduce memory usage and is mainly for testing purposes.

Example:

```
Solver = ELPA {
  Mode = 2
  Autotune = Yes
  Gpu = Yes
}
```

One caveat for this solver is that the number of parallel groups (see p. 100) must match the number of k-points (times 2 in the case of collinear spin polarisation). Calculations without k-points can use either one or two groups in the case of collinear spin polarisation.

OMM

This method minimises the single particle density matrix, so does not make band structure information available. It is only stable for insulating grounds states, i.e., systems with a HOMO-LUMO (band) gap.

The orbital minimisation method has four options:

nInterationsELPA	i	5
Tolerance	r	1E-10
Choleskii	l	Yes
Sparse	l	No

nInterationsELPA Number of initial iterations to be performed with ELPA before the OMM method starts.

Tolerance Minimisation tolerance for this solver, larger values are faster by may be less stable.

Choleskii Whether the overlap is Choleskii factorised before applying OMM. This may increase stability of this method.

Sparse Whether the code should use the sparse matrix interface to ELSI solvers. This does not substantially improve memory usage in this case as internally the dense problem is solved with libOMM.

PEXSI

The PEXSI solver directly calculates the density matrix, so does not make band structure information or Mermi free energy available. The scaling with system size is better than the other solvers available in DFTB+, increasing as $O(N_{\text{atom}}^{d/2+1/2})$ where d is the effective dimensionality of the system. Hence for three dimensional structures it will scale as $O(N^2)$ for general systems.

Method	i	(PEXSI $>$ 2.5)	3
Poles	i	Method = 3	30
ProcsPerPole	i		1
muPoints	i		2
SymbolicFactorProcs	i		1
SpectralRadius	r		10
Sparse	l		No
Threshold	l	Sparse = No	1E-15

Method used for pole expansion. ELSI v2.5 only supports methods 1 and 2, but v2.6 onwards offers method 3 as well (see the ELSI user manual for details of the methods). Methods 1 and 3 can calculate the electron entropy, so enable the Mermin energy to be evaluated. When compiled with ELSI v2.5, the DFTB+ default is Method=2, while for later versions method 3 is the default.

Poles number of poles for the complex plane calculation. Method=1 defaults to 60 poles, 2 has 20 and method 3 uses 30 by default.

ProcsPerPole processors used to calculate the inversion at each pole.

muPoints number of processors used to search for the Fermi level.

SymbolicFactorProcs number of processors to use in evaluating the factorisation pattern of matrices.

SpectralRadius [*Energy*] extension of the complex contour.

Sparse Whether the code should use the sparse ELSI matrix interface.

Threshold Sets the threshold to convert dense matrices to the internal sparse representation that ELSI uses. This may be useful in the case of matrix factorisation issues inside the solver.

NTPoly

This method constructs the single particle density matrix via a purification method based on matrix polynomials (hence requires insulating systems). The solver does not make band structure information available, but can be linear scaling in both time and memory depending on settings and system. Currently the solver does not support spin polarisation or k-points.

This solver has several options:

PurificationMethod	i		2
Tolerance	r		1E-5
Truncation	r		1E-10
Sparse	l		No
Threshold	l	Sparse = No	1E-15

PurificationMethod Allowed choices are 0 for canonical purification, 1 for trace correcting purification, 2 for 4th order trace resetting purification, and 3 for generalised hole-particle canonical purification.

Tolerance Iterative convergence tolerance for this solver, larger values are faster by may be less stable.

Truncation Tolerance below which matrix elements in the density matrix are dropped to enforce sparsity.

Sparse Whether the code should use the sparse matrix ELSI interface.

Threshold Sets the threshold to convert dense matrices to the internal sparse representation that NTPoly uses.

The default choices of Tolerance and Truncation lead to an accurate, but slow, solutions. Alternatively linear scaling can be achieved at smaller system sizes with a larger choice of these values. Values in the range of 1E-3 and 1E-6 for Tolerance and Truncation may be suitable (but test the quality of the solutions).

2.4.8 Filling

There are currently two types of filling supported (see below). Both have common options:

Temperature	r	AdaptFillingTemp = No	0.0
IndependentKFilling	l	Periodic = Yes	No
FixedFermiLevel	(1|2)r		-

Temperature [*energy*] Electron temperature in energy units. This property is ignored for thermostated MD runs, if the AdaptFillingTemp property of the thermostat has been set to Yes (See p. 23).

IndependentKFilling Causes the occupation of the eigenstates to be independently determined for each k-point, thus preventing electron transfer between the k-points. Please note that the value for the Fermi level printed out by the code is meaningless in that case, since there is no common Fermi level for all k-points. This option is incompatible with use of the FixedFermiLevel keyword.

FixedFermiLevel [*energy*] Can be used to fix the Fermi-level (total chemical potential, μ) of the electrons in the system. For collinear spin polarisation, values for up and down spin channels are required. Otherwise only a single global chemical potential is required. If this option is present, the total charge and the total spin of the system are not conserved (settings in the options Charge and UnpairedElectrons will be ignored). If a fixed chemical potential is used, the output *force related energy* includes the contribution to the free energy, $-N\mu$, hence if differentiated will give the forces and stresses (if periodic).

Fermi{}

Fills the single particle levels according to a Fermi distribution. When using a finite temperature, the Mermin free energy (which the code prints) should be used instead of the total energy. This is given by $E - TS$, where the electron entropy S is used.

Example:

```
Filling = Fermi {
  Temperature [K] = 300
}
```

MethfesselPaxton{}

Produces a Fermi-like distribution but with much lower electron entropy [62]. This is useful for systems that require high electron temperatures (for example when calculating metallic systems). There is an additional option for this type of filling:

Order	i	2

Order Order of the Methessel-Paxton scheme, the order must be greater than zero, and the 1st order scheme is equivalent to Gaussian filling.

Note: Due to the non-monotonic behaviour of the Methfessel-Paxton filling function, the position of the Fermi-level is not necessary unique for a given number of electrons. Therefore, different fillings, band entropies, and Mermin free energies may result, depending which one has been found by the Fermi-level search algorithm. The differences, however, are usually not physically significant.

2.4.9 Time-independent DFTB (TI-DFTB) excited states

ΔDFTB

The time-independent approach ΔDFTB modifies the SCC loop to allow variational relaxation of the lowest singlet excited state of closed-shell singlet species.[48] The KS spin orbitals are self-constently optimised under non-Aufbau orbital occupation constraints via promotion of an electron from the highest occupied molecular orbital (HOMO) of the ground state to lowest unoccupied molecular orbital (LUMO) at each SCC iteration. Specifically, the NonAufbau{} block in DFTB+manipulates the electronic filling to emulate a singlet excited state. By modifying the occupation pattern, the target excited state density (ρ_e) is can be constructed according to the standard formula for obtaining the electron density from the occupied colinear spin orbitals,

$$\rho_e(r) = \sum_{\sigma} \sum_{i \varepsilon occ(\sigma)} \mid \phi_i^\sigma(r) \mid^2 .$$

Because the constraint is placed on the spin orbitals rather than the MO's, this method introduces heavy spin contamination to the system by the arbitrary promotion of α spin electrons. This issue is circumvented with spin purification, where the triplet state energy is subtracted from twice the mixed singlet state energy, in accordance with the Ziegler sum rule,[97]

$$E(S_1) = 2E[\{\phi_i^\sigma\}_m] - E[\{\phi_i^\sigma\}_t].$$

A non-spin-purified calculation can be performed if SpinPurify = No. ΔDFTB calculations can be more challenging to converge than their ground-state analogs. To aid convergence, one can first run a ground-state initial guess with GroundGuess = Yes. The DIIS mixer is often an effective alternative mixer for converging ΔDFTB calculations in cases where the calculation fails to converge with the Broyden mixer.

This is a singlet excitation from a spin zero ground state system, so SpinPolarisation should not be set in the input for this type of calculation, but spin constants (p. 43) *are* required. The calculation can be modified by including the following properties in the nonAufbau{} block:

SpinPurify	l	Yes
GroundGuess	l	No

SpinPurify Calculates both a triplet and mixed state for spin-purification. If set to No, only the mixed state is calculated. The mixed state is formed by promoting an α electron from the HOMO to the LUMO.

GroundGuess Calculates the ground state energy prior to a non-Aufbau calculation. The SCC loop will run three times if NonAubau, SpinPurify, and GroundGuess are set to Yes, but only twice if SpinPurify = No.

The default settings, if NonAufbau{} is included in the input, is equivalent to:

```
NonAufbau = {
  GroundGuess = No
  SpinPurify = Yes
}
```

This performs two self-consistent calculations at each geometry step, and then evaluates the energy, forces and charges in the ΔDFTB first excited state. Note that this is a singlet state, hence only the *total* charge is relevant in Mulliken populations.

If instead the ground state is also requested (SpinPurify = Yes by default):

```
NonAufbau = {
  GroundGuess = Yes
}
```

The detailed.out file will then contain energies of the ground, triplet and the spin purified singlet excitation, along with estimates for the excitation energies from the S_o to the T_1 and S_1 states:

```
S0 -> T1:                0.2104101411 H        5.7256 eV
S0 -> S1:                0.2615036445 H        7.1159 eV
```

2.4.10 SlaterKosterFiles

There are two different ways to specify the Slater-Koster files for the atom type pairs, explicit specification and using the Type2FileNames{} method.

All specified Slater-Koster files are additionally searched for in directory specified in the DFTB-PLUS_PARAM_DIR environment variable.

Explicit specification

Every pairwise permutation atomic types, connected by a dash, must occur as a property with the name of the corresponding file as an assigned value.

Example (GaAs):

```
SlaterKosterFiles = {
  Ga-Ga = "./Ga-Ga.skf"
  Ga-As = "./Ga-As.skf"
  As-Ga = "./As-Ga.skf"
  As-As = "./As-As.skf"
}
```

If you treat shells from different species as shells of one atom by using the SelectedShells{} keyword in the MaxAngularMomentum{} block, you have to specify more than one file name for certain species pairs. (For details see the description about the MaxAngularMomentum{} keyword.)

Type2FileNames{}

You can use this method to generate the name of the Slater-Koster files automatically using the element names from the input geometry. You have to specify the following properties:

Prefix	s	""
Separator	s	""
Suffix	s	""
LowerCaseTypeName	l	No

Prefix Prefix before the first type name, usually the path.

Separator Separator between the type names.

Suffix Suffix after the name of the second type, usually extension.

LowerCaseTypeName If the name of the types should be converted to lower case. Otherwise they are used in the same way, as they were specified in the geometry input.

Example (for producing the same file names as in the previous section):

```
SlaterKosterFiles = Type2FileNames {
  Prefix = "./"
  Separator = "-"
  Suffix = ".skf"
  LowerCaseTypeName = No
}
```

The Type2FileNames method can not be used if an extended basis was defined with the Selected-Shells method.

2.4.11 KPointsAndWeights

The k-points for the Brillouin-zone integration can either be specified explicitly, or automatically for supercells by using the SupercellFolding{} or KLines{} methods for supercells or either Heli-calUniform{} or HelicalSampled{} for helical boundary conditions.

Note: For the automatic grid methods, the KPointsAndWeights keyword is not allowed to have a modifier.

Explicit specification

Each k-point and its weight in the integral should be specified, for supercells this requires that four real numbers must be specified for each point: The coordinates of the given k-point followed by its weight, while for helical coordinates there are two coordinates (along the helical axis and with respect to the rotation around the axis) and the weight of the point. By default, coordinates are specified in fractions of the reciprocal lattice vectors. If the modifier absolute is set for the KPointsAndWeights keyword, absolute k-point coordinates in atomic units are instead expected. The sum of the k-point weights is automatically normalised by the program.

```
KPointsAndWeights = {   # 2x2x2 MP-scheme
  0.25  0.25  0.25    1.0
  0.25  0.25 -0.25    1.0
  0.25 -0.25  0.25    1.0
  0.25 -0.25 -0.25    1.0
}
```

SupercellFolding{}

This method generates a sampling set containing all the special k-points in the Brillouin zone related to points that would occur in an enlarged supercell repeating of the current unit cell. If two k-points in the BZ are related by inversion, only one (with double weight) is used (in the absence of spin-orbit coupling this is permitted by time reversal symmetry). The SupercellFolding{} method expects 9 integers and 3 real values as parameters:

$$\begin{array}{ccc} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \\ s_1 & s_2 & s_3 \end{array}$$

The integers n_{ij} specify the coefficients used to build the supercell vectors $\mathbf{A}_i$ from the original lattice vectors $\mathbf{a}_j$:

$$\mathbf{A}_i = \sum_{j=1}^{3} n_{ij}\,\mathbf{a}_j.$$

The real values, s_i, specify the point in the Brillouin-zone of the super lattice, in which the folding should occur. The coordinates must be given in relative coordinates, in the units of the reciprocal lattice vectors of the super lattice.

The original $l_1 \times l_2 \times l_3$ Monkhorst-Pack sampling [64] for cubic lattices corresponds to a uniform extension of the lattice:

$$\begin{array}{ccc} l_1 & 0 & 0 \\ 0 & l_2 & 0 \\ 0 & 0 & l_3 \\ s_1 & s_2 & s_3 \end{array}$$

where s_i is 0.0, if l_i is odd, and s_i is 0.5 if l_i is even. For the $2 \times 2 \times 3$ scheme, you would write for example

```
# 2x2x3 MP-scheme according original paper
KPointsAndWeights = SupercellFolding {
  2   0   0
  0   2   0
  0   0   3
  0.5  0.5  0.0
}
```

To use k-points for hexagonal lattices which are consistent with the erratum to the original paper [65], you should set the shift for the unique "c" direction, s_3, in the same way as in the original scheme. The s_1 and s_2 shifts should be set to be 0.0 independent of whether l_1 and l_2 are even or odd. So, for a $2 \times 3 \times 4$ sampling you would have to set

```
# 2x3x4 MP-scheme according modified MP scheme
KPointsAndWeights = SupercellFolding {
  2   0   0
  0   3   0
  0   0   4
  0.0  0.0  0.5
}
```

It is important to note that DFTB+ does not take the symmetry of your system explicitly into account. For small high symmetric systems with a low number of k-points in the sampling this could eventually lead to unphysical results. (Components of tensor properties–e.g. forces–could be finite, even if they must vanish due to symmetry reasons.) For those cases, you should explicitly specify k-points with the correct symmetry.

KLines{}

This method specifies k-points lying along arbitrary lines in the Brillouin zone. This is useful when calculating the band structure for a periodic system. (In that case, the charges should be initialised from the saved charges of a previous calculation with a proper k-sampling. Additionally for SCC calculations the number of SCC cycles should be set to 1, so that only one diagonalisation is done using the initial charges.)

The KLines{} method accepts for each line an integer specifying the number of points along the line segment, and 3 real values specifying the end point of the line segment. The line segments do not include their starting points but their end points. The starting point for the first line segment can be set by specifying a (zeroth) segment with only one point and with the desired starting point as end point. The unit of the k-points is determined by any modifier of the KPointsAndWeights property. (Default is relative coordinates.)

Example:

```
KPointsAndWeights [relative] = KLines {
  1  0.5  0.0  0.0    # Setting (and calculating) starting point 0.5 0.0 0.0
 10   0.0  0.0  0.0    # 10 points from 0.5 0.0 0.0  to  0.0 0.0 0.0
 10   0.5  0.5  0.5    # 10 points from 0.0 0.0 0.0 to 0.5 0.5 0.5
  1  0.0  0.0  0.0    # Setting (and calculating) a new starting point
 10   0.5  0.5  0.0    # 10 points from 0.0 0.0 0.0 to 0.5 0.5 0.0
}
```

Note: Since this set of k-points probably does not correctly integrate the Brillouin zone, the default value of MaxSccIterations is set to be 1 in this case.

HelicalUniform{}

This method specifies k-points lying along the generalised reciprocal vector of the Brillouin zone of a helical cell and around the order-n rotational axis (currently the k-points that exactly represent the C_n rotation are used). The HelicalUniform{} method expects 1 integer and 1 real value as parameters, where the first value specifies the number of sampling points along the helical axis, while the second gives the shift (analogous to the three dimensional case of SupercellFolding{}) in this direction. A shift of 0.5 appears to give more rapid convergence of the grid.

Example:

```
KPointsAndWeights = HelicalUniform {80 0.5}
```

HelicalSampled{}

Instead of exactly integrating around the C_n rotation in k-space, the HelicalSampled{} method allows for a sampled integration. It expects 4 values: the first two are the number of sample points along the helix and around the rotation respectively, while the second two are shifts in the grid.

Example:

```
KPointsAndWeights = HelicalSampled {20 4 0.5 0.25}
```

There are several things to note here: firstly, the second grid (values 4 and 0.25 in the example above) approximates the integration around the C_n symmetry, so should only be used where the order of this rotation axis is large. Secondly, non-zero shifts in the grid, particularly for small numbers of sampling points, are likely to be unphysical as are shifts for the grids at the exact order of the rotation operation.

2.4.12 OrbitalPotential

Currently the FLL (fully localised limit) and pSIC [39] (pseudo self interaction correction) forms of the LDA+U corrections [75] are implemented. These potentials effect the energy of states on designated shells of particular atoms, usually increasing the localisation of states at these sites. The FLL potential lowers the energy of occupied states localised on the specified atomic shells while raising the energy of unoccupied states. The pSIC potential corrects the local part of the self-interaction error and so lowers the energy of occupied states (see Ref. [39] for a discussion of the relation between these two potentials, and possible choices for the UJ constant). These particular corrections are most useful for lanthanide/actinide f states and some localised d states of transition metals (Ni3d for example).

The Functional option chooses which correction to apply, followed by a list of the specific corrections, listed as an atomic species and the shells on that atom followed by the $U-J$ constant for that block of shells.

```
OrbitalPotential = {
 Functional = {FLL}
 Si = {
   Shells = {1 2} # sp block on the atom
   UJ = 0.124
 }
}
```

2.4.13 ElectricField

This tag contains the specification for an external electric field. Electric fields can only be specified for SCC calculations. You can apply the electric field of point charges[4] and/or a homogeneous external field (which may change harmonically in time). The ElectricField block can currently contain either one or more PointCharges blocks and potentially an External block.

PointCharges

The specification for PointCharges has the following properties:

CoordsAndCharges	(4r)+		-
GaussianBlurWidth	r	Periodic = No	0.0

CoordsAndCharges [*length*] Contains the coordinates and the charge for each point charge (four real values per point charge). A length modifier can be used to alter the units of the coordinates. The charge must be specified in proton charges. (The charge of an electron is -1.)

If you read in a huge number of external charges the parsing time to process this data could be unreasonably long. You can avoid this by including the coordinates and the charges directly

[4]Only in calculations with fixed lattice constants.

from an external file via the DirectRead{} method (see the example in the next paragraph). Please note that when using this method the program will only read the specified number of records from the external file, and ignores any additional data (so do not leave comments in the external file for example). The external file should contain only one record (3 coordinates and 1 charge) per line.

GaussianBlurWidth [*length*] Specifies the half width σ of the Gaussian charge distribution, which is used to delocalise the point charges. The energy of the coulombic interaction E_C between the delocalised point charge M with charge Q_M and the atom A with charge q_A is weighted by the error function as

$$E_C(A,M) = \frac{q_A Q_M}{r_{AM}} \operatorname{erf}\left[\frac{r_{AM}}{\sigma}\right],$$

where r_{AM} is the distance between the point charge and the atom.

A length modifier can be used to specify the unit for σ.

Example:

```
ElectricField = {
  # 1st group of charges, with Gaussian delocalisation
  # We have 100000 charges, therefore we choose the fast reading method.
  PointCharges = {
    GaussianBlurWidth [Angstrom] = 3.0
    CoordsAndCharges [Angstrom] = DirectRead {
      Records = 100000
      File = "charges.dat"
    }
  }
  # 2nd group of charges, no delocalisation (sigma = 0.0)
  PointCharges = {
    CoordsAndCharges [Angstrom] = {
      3.3  -1.2  0.9     9.2
      1.2  -3.4  5.6    -3.3
    }
  }
}
```

Note: External charges are currently only available for the DFTB (section 2.4.1) hamiltonian, and not for the xTB model (section 2.4.2).

External

Specifies a homogeneous external electric field. In the case of *periodic* calculations, a saw-tooth potential is currently used, hence it is up to the user to guarantee that there is a vacuum region isolating periodic copies of the system along the applied field direction. We suggest that you place the structure in the 'middle' of the unit cell if possible, to reduce the chances of atoms approaching cell boundaries along the direction of the applied electric field. The code will halt if atoms interact with periodic images of the unit cell along the direction of the electric field.

The External field keyword has the following options

Strength	r		-
Direction	3r		
Frequency	r	molecular dynamics used	0.0
Phase	i	Geometry step offset	0

Strength [*Electric field strength*] Specified strength of the applied field.

Direction Vector direction of the applied field (the code normalises this vector). In the case of periodic calculations, currently the system *must not* be continuous in this direction (see above).

Frequency [*Frequency*] If using molecular dynamics, the field can be time varying with this frequency.

Phase Initial field phase in units of geometry steps, this is needed if restarting an MD run in an external field to give the offset in phase of the field after the specified number of steps from the old calculation. The applied field is of the form

$$\mathbf{E}_0 \sin(\omega \Delta t (step + phase))$$

where $\mathbf{E}_0$ is the field vector specified by Strength and Direction, ω is the angular Frequency and $step$ is the current MD-step in the simulation, using the MD TimeStep of Δt (see section 2.3.8).

2.4.14 AtomSitePotential

Specifies an external potential at a selected atomic site or sites. This potential either couples with the gross atomic charges (charges printed in detailed.out when MullikenAnalysis = Yes) or net atomic charge (as printed when WriteNetCharges is enabled). **Note:** The sign convention is that the potential will *increase* the energy of states associated with the affected site for a *positive* potential.

The Net and Gross potentials are specified as environments, both of which have the following options

Atoms	i+
Vext	r+

Atoms The atoms in the geometry for which an external potential should be applied. As these must be paired with a specified potential value, specific atom numbers in the structure should be given (not ranges or chemical species).

Vext [*energy*] Specified strength of the applied potential at the listed sites.

Example:

```
AtomSitePotential {
  # potential(s) coupling to Mulliken charges
  Gross {
    Atoms = 1 2
    Vext [eV] = 1.5 -0.5
  }
  # on-site only part of potential
  Net {
    Atoms = 2
    Vext [eV] = 0.5
  }
}
```

2.4.15 Dispersion

The Dispersion block controls whether DFTB interactions should be empirically corrected for van der Waals interactions, since DFTB (and SCC-DFTB) does not include these effects. Currently, six different dispersion correction schemes are implemented (for the detailed description of the methods see the following subsections):

- LennardJones: Dispersion is included via a Lennard-Jones potential between each pair of atoms. The parameters for the potential can either be entered by the user or the program can automatically take the parameters from the Universal Force Field (UFF) [78].
- SlaterKirkwood: The dispersion interaction between atoms is taken from a Slater-Kirkwood polarisable atomic model [22].
- DftD3: Dispersion is calculated as in the s-dftd3 library [29, 30] (see section 2.4.15). Modification hydrogen bond interaction strengths (see section 2.4.19). Note: If DFTB+ is compiled without the s-dftd3 library, the SimpleDftD3 can be used to include D3(BJ)-dispersion.
- SimpleDftD3: DFTB+ internal implementation of the D3-dispersion model.
- DftD4: Dispersion is calculated using the D4 model [15, 16] (see section 2.4.15).
- Ts: Dispersion is calculated using the Tkatchenko-Scheffler model [89] adapted for DFTB [87] (see section 2.4.15).
- Mbd Dispersion is calculated using the MBD@rsSCS model [3] adapted for DFTB [87] (see section 2.4.15).

LennardJones

The Lennard-Jones dispersion model in DFTB+ follows the method of Ref. [96], using the following potential:

$$
\begin{aligned}
U_{ij}(r) &= d_{ij}\left[-2\left(\frac{r_{ij}}{r}\right)^6 + \left(\frac{r_{ij}}{r}\right)^{12}\right] \qquad r >= r_0 \\
U_{ij}(r) &= U_0 + U_1 r^5 + U_2 r^{10} \qquad r < r_0
\end{aligned}
$$

where r_0 is the distance at which the potential turns from repulsive to attractive. The parameters d_{ij} and r_{ij} are built from atomic parameters d_i, d_j and r_i, r_j via the geometrical mean ($d_{ij} = \sqrt{d_i d_j}$, $r_{ij} = \sqrt{r_i r_j}$). The parameters U_0, U_1, U_2 ensure a smooth functional form at r_0.

The parameters r_i and d_i can either be taken from the parameters of the UFF [78] (as in Ref. [96]) or can be specified manually for each species.

Example using UFF parameters:

```
Dispersion = LennardJones {
  Parameters = UFFParameters {}
}
```

Example using manually specified parameters:

```
Dispersion = LennardJones {
  Parameters {
   H {
     Distance [AA] = 2.886
     Energy [kcal/mol] = 0.044
   }
   O {
     Distance [AA] = 3.500
     Energy [kcal/mol] = 0.060
   }
  }
}
```

The UFF provides dispersion parameters for nearly every element of the periodic table, therefore it can be used for almost all systems "out of the box". The parameters are also independent of the atomic coordination number, allowing straight forward geometry relaxation or molecular dynamics (unlike the current implementation of Slater-Kirkwood type dispersion).

SlaterKirkwood

A Slater-Kirkwood type dispersion model is also implemented in DFTB+ as described in Ref. [22].[5] This model requires atomic polarisation values, van der Waals radii and effective charges for every atom in your system. These parameters are dependent on the coordination of each atom, hence values for different atoms of the same species may vary depending on local environment. You can supply these parameters for the atoms in either of two ways, both using the PolarRadiusCharge tag.

The first option is to specify the values within the PolarRadiusCharge environment by providing three real values (polarisability, van der Waals radius, effective charge) for each atom separately.

Example:

```
Dispersion = SlaterKirkwood {
  # Using Angstrom^3 for volume, Angstrom for length and default
  # unit for charge (note the two separating commas between the units)
  PolarRadiusCharge [Angstrom^3,Angstrom,] = {
    # Polar      Radius      Chrg
    0.560000    3.800000    3.150000      # Atom 1: O
    0.386000    3.500000    0.800000      # Atom 2: H
    0.386000    3.500000    0.800000      # Atom 3: H
  }
}
```

[5]Please note, that Ref. [22] contains two typos: equation (7) should read $C_6^{\alpha\beta} = \frac{2C_6^{\alpha}C_6^{\beta}p_\alpha p_\beta}{p_\alpha^2 C_6^{\beta} + p_\beta^2 C_6^{\alpha}}$, in equation (9) the contribution from the dispersion should be $E_{\text{dis}} = -\frac{1}{2}\sum_{\alpha\beta} f(R_{\alpha\beta}) C_6^{\alpha\beta} (R_{\alpha\beta})^{-6}$.

Alternatively you can provide values for each atomic species in your system, but must supply different values for different coordination numbers using the HybridDependentPol{} keyword. The code needs specific parameters for each type of atom in environments with 0, 1, 2, 3, 4 or ⩾5 neighbours. DFTB+ then picks the appropriate values for each atom based on their coordination in the *starting* geometry. Two atoms are considered to be neighbours if their distance is less than the sum of their covalent radii, hence you need to supply the covalent radii for each atomic species using the CovalentRadius keyword. This is then followed by a HybridPolarisations / HybridPolarization block containing a list of six values for atomic polarisabilities then six van der Waals radii and finally a single hybridisation independent effective charge for that atomic species.

Example:

```
Dispersion = SlaterKirkwood {
  PolarRadiusCharge = HybridDependentPol {
    O = {
      CovalentRadius [Angstrom] = 0.8
      HybridPolarisations [Angstrom^3,Angstrom,] = {
        # Atomic polarisabilities 0-5       van der Waals radii 0-5  chrg
        0.560 0.560 0.560 0.560 0.560 0.560  3.8 3.8 3.8 3.8 3.8 3.8  3.15
      }
    }
    H = {
      CovalentRadius [Angstrom] = 0.4
      HybridPolarisations [Angstrom^3,Angstrom,] = {
        # Atomic polarisabilities 0-5       van der Waals radii 0-5  chrg
        0.386 0.396 0.400 0.410 0.410 0.410   3.5 3.5 3.5 3.5 3.5 3.5   0.8
      }
    }
  }
}
```

Warning: For both methods of specifying the Slater-Kirkwood dispersion model the code keeps the dispersion parameters fixed for each atom during the entire calculation. Even if the geometry (and therefore the hybridisation) of atoms changes significantly during the calculation, the parameters are unchanged. Therefore if atoms are able to move during your calculation (geometry relaxation or molecular dynamics) you should *always* check whether the coordination of your atoms has changed during the run.

Furthermore, when using the HybridDependentPol{} method we suggest that you first set the StopAfterParsing keyword in the ParserOptions block to Yes (see p. 100) and inspect the generated polarisabilities, radii and charges for every atom in the dftb_pin.hsd file. If fine tuning of the generated values turns out to be necessary, you should replace the hybrid dependent specification in the input file with corrected atom specific values based on dftb_pin.hsd.

In order to find suitable parameters for the Slater-Kirkwood model, you should consult Ref. [22] and further references therein. Appendix F contains values which have already been used by some DFTB-users for a few elements.

DftD3

Note: the DFTB+ binary must be compiled with the DFT-D3 library enabled to use this feature. It is recommended to use the SimpleDftD3 dispersion model instead, which offers similar functionality but without any library dependence.

The DFT-D3 dispersion correction in DFTB+ is an implementation of the method used in the code 'dftd3' by Stefan Grimme and coworkers. It is based on the *ansatz* described in Refs. [29] and [30].

This dispersion correction for DFTB adds a contribution to the general Kohn-Sham-like energy

$$E_{\text{DFTB-D3}} = E_{\text{DFTB}} + E_{\text{disp}}$$

with E_{DFTB} being the DFTB total energy and E_{disp} the dispersion energy. The latter contains two-body and optional three-body contributions:

$$E_{\text{disp}} = E^{(2)}_{\text{disp}} + E^{(3)}_{\text{disp}}.$$

The form of the two-body contribution can change depending on the chosen damping factor:

- Becke-Johnson damping function:

  $$E^{(2)}_{\text{disp}} = -\frac{1}{2}\sum_{A\neq B}\sum_{n=6,8} s_n \frac{C_n^{AB}}{r_{AB}^n + f(R_0^{AB})}$$

  with

  $$f(R_0^{AB}) = a_1 R_0^{AB} + a_2.$$

- Zero-damping (dispersion at short distances is damped to zero):

  $$E^{(2)}_{\text{disp}} = -\frac{1}{2}\sum_{A\neq B} s_n \frac{C_n^{AB}}{r_{AB}^n} f_{d,n}(r_{AB})$$

  with

  $$f_{d,n} = \frac{1}{1+6(r_{AB}/(s_{\text{r},n}R_0^{AB}))^{-\alpha_n}}$$

- Modified zero-damping (dispersion at short distances is damped to zero):

  $$E^{(2)}_{\text{disp}} = -\frac{1}{2}\sum_{A\neq B} s_n \frac{C_n^{AB}}{r_{AB}^n} f_{d,n}(r_{AB})$$

  with

  $$f_{d,n} = \frac{1}{1+6(r_{AB}/(s_{\text{r},n}R_0^{AB}+\beta))^{-\alpha_n}}$$

In order to adjust the dispersion for various energy functionals, the choice of s_6, s_8 and the damping parameters a_1 and a_2 (for Becke-Johnson-damping), $s_{\text{r},6}$ and α_6 (for zero damping) or $s_{\text{r},6}$, α_6 and β (for modified zero damping) are treated as functional-dependent values. All other parameters are fixed based on these parameters.

As the DFTB energy functional is largely determined by the underlying parameterisation (the Slater-Koster-files) and the chosen DFTB model (e.g. non-scc, scc, 3rd order, etc.), there are no universal parameter choices which can be used with all settings, but some relevant choices for various parameterisation are given in Appendix G.

Note: for the version 6 or earlier of the DFTB+ input parser (see section 2.10) the default values of these parameters are set to be appropriate for DFTB3. But from parser version 7 onwards, no default values are set.

Example using adjusted parameters with Becke-Johnson damping:

```
Dispersion = DftD3 {
  Damping = BeckeJohnson {
    a1 = 0.5719
    a2 = 3.6017
  }
  s6 = 1.0
  s8 = 0.5883
}
```

Example using zero-damping:

```
Dispersion = DftD3 {
  Damping = ZeroDamping {
    sr6 = 0.7461
    alpha6 = 14.0
  }
  s6 = 1.0
  s8 = 3.209
}
```

DftD3 optional settings

Apart from the functional dependent dispersion parameters, you can also adjust the additional parameters as shown below. The default values for these parameters are taken to be the same as in the dftd3 code.

Cutoff	r	$\sqrt{9000}$
CutoffCN	r	40
Threebody	l	No
HHRepulsion	l	No
AtomicNumbers	m	{}

Cutoff [*length*] Cutoff distance when calculating two-body interactions.

CutoffCN [*length*] Cutoff distance when calculating three-body interactions.

Threebody Whether three-body contributions should be included in the dispersion interactions.

AtomicNumbers Allows overwrite of atomic numbers associated with a species name.

```
# Default species (H-Og) are implied
AtomicNumbers {
  D = 1  # set deuterium as hydrogen
  S = 6  # overwrite sulfur as carbon
}
```

HHRepulsion Required when calculating the DFTB3-D3H5 [80] modification to D3 dispersion (see section 2.4.19 for details and parameter values). This keyword enables an additional short range repulsion term in all hydrogen–hydrogen pairs [81] which prevents them from approaching too closely together.

SimpleDftD3

D3-dispersion with Becke-Johnson damping. See section 2.4.15 for details and references.

s6	r	1.0	
s8	r		
s10	r	0.0	
a1	r		
a2	r		
alpha	r	16.0	
WeightingFactor	r	6.0	
CutoffInter	r	64	
CoordinationNumber	m	Exp	65

s8, s9, a1, a2 Functional dependent dispersion parameters, see equations above.

s6 Parameter for scaling pairwise dipole–dipole dispersion interaction, should always be set to 1.0.

s10 Parameter for pairwise quadrupole–quadrupole dispersion interactions, should be kept set to 0.0.

alpha Zero damping exponent for three-body damping function.

WeightingFactor Coordination number based interpolation.

CutoffInter [*length*] Cutoff distance when calculating two-body interactions.

CoordinationNumber Method to determine the coordination numbers. See 65 for details.

Example using adjusted parameters:

```
Dispersion = SimpleDftD3 {
  a1 = 0.5719
  a2 = 3.6017
  s6 = 1.0
  s8 = 0.5883
}
```

DftD4

The DFT-D4 dispersion correction in DFTB+ is an implementation of the D4 model by Stefan Grimme and coworkers. It is based on the method described in Ref. [16].

This dispersion correction for DFTB adds a contribution to the general Kohn–Sham-like energy,

$$E_{\text{DFTB-D4}} = E_{\text{DFTB}} + E_{\text{disp}},$$

with E_{DFTB} being the DFTB total energy and E_{disp} the dispersion energy. The latter contains two-body and optional three-body contributions:

$$E_{\text{disp}} = E^{(2)}_{\text{disp}} + E^{(3)}_{\text{disp}}.$$

The D4 model uses the Becke–Johnson damping function for two-body contributions:

$$E^{(2)}_{\text{disp}} = -\frac{1}{2}\sum_{A\neq B}\sum_{n=6,8,10} s_n \frac{C_n^{AB}}{r_{AB}^n + f(R_0^{AB})}$$

with

$$f(R_0^{AB}) = a_1 R_0^{AB} + a_2.$$

The zero-damping function for three-body contributions is:

$$E^{(3)}_{\text{disp}} = -\sum_{A}\sum_{\substack{B\\B<A}}\sum_{\substack{C\\C<B}} s_9 \frac{(3\cos\theta_A\cos\theta_B\cos\theta_C + 1)\sqrt{C_6^{AB}C_6^{BC}C_6^{CA}}}{(r_{AB}r_{BC}r_{CA})^3} f^{(3)}(r_{AB})$$

with

$$f^{(3)} = \frac{1}{1+6\left(\frac{f(R_0^{AB})f(R_0^{BC})f(R_0^{CA})}{r_{AB}r_{BC}r_{CA}}\right)^{\alpha/3}}.$$

In order to adjust the dispersion for various energy functionals, the choice of s_8 and the damping parameters a_1 and a_2 are treated as functional-dependent values. All other parameters are fixed based on these parameters. Depending on the choice of the s_9 parameter non-additive triple-dipole contributions will be evaluated. Including non-additive effects usually improves the description of dispersion interactions, but is also more expensive.

As the DFTB energy functional is largely determined by the underlying parameterisation (the Slater–Koster-files) and the chosen DFTB model (e.g. non-scc, scc, 3rd order, etc.), there are no universal parameter choices which can be used with all settings, but some relevant choices for various parameterisation are given in Appendix H.

DftD4 settings

Beside the functional dependent dispersion parameters, the options shown below can be adjusted in the input.

s6	r	1.0	
s8	r		
s10	r	0.0	
s9	r		
a1	r		
a2	r		
alpha	r	16.0	
WeightingFactor	r	6.0	
ChargeSteepness	r	2.0	
ChargeScale	r	3.0	
CutoffInter	r	64	
CutoffThree	r	40	
CoordinationNumber	m	Cov	65
ChargeModel	m	EEQ or SelfConsistent	64
AtomicNumbers	m	{}	

s8, s9, a1, a2 Functional dependent dispersion parameters, see equations above.

s6 Parameter for scaling pairwise dipole–dipole dispersion interaction, should always be set to 1.0.

s10 Parameter for pairwise quadrupole–quadrupole dispersion interactions, should be kept set to 0.0.

alpha Zero damping exponent for three-body damping function, default is 16 as in DFT-D3.

WeightingFactor Coordination number based interpolation, 4.0 used in DFT-D3, default for D4 is 6.0.

ChargeScale Maximum possible charge scaling, used as exponential value, should be kept set to 3.0.

ChargeSteepness Steepness of the charge scaling function, should be kept set to 2.0.

CutoffInter [*length*] Cutoff distance when calculating two-body interactions.

CutoffThree [*length*] Cutoff distance when calculating three-body interactions.

AtomicNumbers Allows overwrite of atomic numbers associated with a species name.

DftD4 ChargeModel

This implementation of DFT-D4 supports the EEQ{} method to initialise the charge model with an electronegativity equilibration (EEQ) model[79] as well as a self-consistent evaluation of the dispersion with SelfConsistent{}.

For the electronegativity equilibration model the following parameters are available. For each species four parameters (Chi, Gam, Kcn, and Rad) have to be supplied in a Values{} method, since the model is instanciated inside the DftD4{} method, Defaults{} for all elements up to 86 can be supplied automatically.[16]

Chi	m	Defaults	
Gam	m	Defaults	
Kcn	m	Defaults	
Rad	m	Defaults	
Cutoff	r	40	
EwaldParameter	r	0.0	
EwaldTolerance	r	1.0e-9	
CoordinationNumber	m	Erf	65

Chi Electronegativities of all species.

Gam Chemical hardnesses of all species.

Kcn CN scaling factor of all species.

Rad Charge width of all species in Bohr.

Cutoff [*length*] Cutoff distance when calculating electrostatics interactions under PBC.

EwaldParameter Sets the splitting parameter in the Ewald electrostatic summation for periodic calculations. This controls the fraction of the Ewald summation occurring in real and reciprocal space. Setting it to zero or negative activates an automatic determination of this parameter (default and recommended behaviour).

EwaldTolerance Sets the tolerance for Ewald summation in periodic systems.

```
ChargeModel = EEQ {
  EwaldParameter = 0.25165824
  EwaldTolerance = 1.0E-9
  Chi = Values {
    Ga = 1.15018618
    As = 1.36313743
  }
  Gam = Values {
    Ga = 8.299615E-2
    As = 0.19005278
  }
  Kcn = Values {
    Ga = -1.05627E-002
    As = 7.657769E-002
  }
  Rad = Values {
    Ga = 1.76901636
    As = 2.41244711
  }
}
```

DftD4 CoordinationNumber

The CoordinationNumber determines how the local coordination environment for its parent method is calculated. Currently four different counting functions are available: Erf{}, Cov{}, Gfn{}, and Exp{}. Erf{} is the default coordination number for the EEQ charge model, while Cov{} is the default coordination number for DFTD4.

Electronegativities	m	PaulingEN
Radii	m	CovalentRadiiD3
Cutoff	r	40
CutCN	r	0 / 8

Radii Covalent radii of all species in Bohr. Default values taken are the DFTD3 covalent radii.[29]

Electronegativities Electronegativities of all species. Default values taken are Pauling ENs.

Cutoff [*length*] Cutoff distance when evaluating counting function.

CutCN Maximum value for coordination number, coordination numbers higher than this value will be smoothly cut away. Deactivated for values smaller or equal to zero. Default depends on parent method.

```
CoordinationNumber = Cov {
  CutCN = 0
  Electronegativities = PaulingEN {}
  Radii = CovalentRadiiD3 {}
}
```

TS

Applies the Tkatchenko-Scheffler dispersion model [89, 87].

The following keywords can be used with this dispersion model:

Damping	r	20.0
RangeSeparation		0.94
ReferenceSet		"TS"

Damping Damping factor.

RangeSeparation Range separation parameter. The default value is the DFTB LDA-value. Note, that various DFTB-parametrisations may require different values.

ReferenceSet Reference values for the free atoms. Possible choices are TS (mostly used for molecules) and TSsurf (mostly used when calculating surfaces).

MBD

Applies the Tkatchenko many-body-dispersion model [3, 87].

The following keywords can be used with this dispersion model:

Beta	r	0.83
NOmegaGrid	i	15
KGrid	3r	
KGridShift		0.5 0.5 0.5
VacuumAxis		No No No
ReferenceSet		"TS"

Beta Range separation parameter. Default value is optimised for pure DFT LDA functionals. The value may be different for various SK-sets.

NOmegaGrid [*force*] Number of integration points.

KGrid Monkhorst-Pack-like k-point grid to integrate the Hamiltonian of the interacting dipoles over the Brillouin-zone. Note, that this grid is independent from the k-point grid used to integrate the atomic Hamiltonian.

KGrid Shift of the Monkhorst-Pack grid.

VacuumAxis Set Yes for each axis, where there is vacuum between the periodically repeated images of the central cell.

ReferenceSet Reference values for the free atoms. Possible choices are TS (mostly used for molecules) and TSsurf (mostly used when calculating surfaces).

2.4.16 DFTB3

If you would like to use what is called “DFTB3” in some publication(s) [27], this group of options include the relevant modifications to the SCC Hamiltonian and energy. *To enable the DFTB3 model* you will need to set ThirdOrderFull = Yes and damp H–X the interactions (see Section 2.4.19).

ThirdOrder If set to Yes the *on-site* 3rd order correction [93] is switched on. This corrects the SCC-Hamiltonian with the derivatives of the Hubbard U parameters, which you have to specify for every element in HubbardDerivs. This correction only alters the on-site elements and is only maintained for backward compatibility. *You should use the full version ThirdOrderFull instead.*

ThirdOrderFull If set to Yes the *full* 3rd order correction [27] is switched on. This corrects the SCC-Hamiltonian with the derivatives of the Hubbard U parameters, which you have to specify for every element in HubbardDerivs.

HubbardDerivs Derivatives of the Hubbard U for the 3rd order correction (on-site or full). For every element the appropriate parameter (in atomic units) must be specified. If you use shell resolved SCC (with full 3rd order), you must specify a list of derivatives for every element, with one Hubbard U derivative for each shell of the given element.

```
Hamiltonian = DFTB {
  :
  ThirdOrder = Yes
  HubbardDerivs {
    O = -0.14
    H = -0.07
  }
  :
}
```

2.4.17 Implicit Solvation Model

Generalised Born Model

In generalised Born (GB) theory,[72] a molecule is considered as continuous region with a dielectric constant ε_{in} surrounded by infinite solvent with a dielectric constant ε_{out}. Charges q_{A} are located at the atomic sites $\vec{R}_{\text{A}}$ and their interaction in the presence of a polarised solvent can be expressed as the solvation energy

$$\Delta G_{\text{GB}} = -\frac{1}{2}\left(\frac{1}{\varepsilon_{\text{in}}} - \frac{1}{\varepsilon_{\text{out}}}\right)\sum_{\text{A}=1}^{N}\sum_{\text{B}=1}^{N}\frac{q_{\text{A}}q_{\text{B}}}{\left(R_{\text{AB}}^2 + a_{\text{A}}a_{\text{B}}\exp\left[-\frac{R_{\text{AB}}^2}{4a_{\text{A}}a_{\text{B}}}\right]\right)^{\frac{1}{2}}}. \tag{2.2}$$

where $a_{\text{A/B}}$ are the effective Born radii of the atoms A/B. The GB model is added to the Hamiltonian as second order fluctuation in the charge density, similar to the coulombic interactions.

The Born radii are evaluated by an Onufriev–Bashford–Case (OBC) corrected pairwise approximation to the molecular volume given as

$$\frac{1}{a_{\text{A}}} = \frac{1}{a_{\text{scale}}}\left(\frac{1}{R_{\text{A}}^{\text{cov}} - R_{\text{offset}}} - \frac{1}{R_{\text{A}}^{\text{cov}}}\cdot\tanh\left[b_{\text{OBC}}\Psi_{\text{A}} - c_{\text{OBC}}\Psi_{\text{A}}^2 + d_{\text{OBC}}\Psi_{\text{A}}^3\right]\right) \tag{2.3}$$

where a_{scale} is a scaling factor for the Born radii, R_{offset} is a global shift parameter for the covalent radii and $a/b/c_{\text{OBC}}$ are the coefficients for the volume polynomial in the OBC correction to the Born radii. Ψ_{A} is the pairwise approximation to the volume integral given by

$$\Psi_{\text{A}} = \frac{R_{\text{A}}^{\text{cov}} - R_{\text{offset}}}{2}\sum_{\text{B}}\Omega(R_{\text{AB}}, R_{\text{A}}^{\text{cov}}, s_{\text{B}}R_{\text{B}}^{\text{cov}}) \tag{2.4}$$

with Ω being the pairwise function used to approximate the volume integral, which is only dependent on the distance and the covalent radii. Note that, the covalent radius of the second atom is scaled by the element-specific descreening value s_B to compensate the systematic overestimation of the volume by this approach.

To use the generalised Born model in the SCC procedure use the GeneralisedBorn{} / GeneralizedBorn{} method in the input to Solvation. The non-polar solvent area model can be combinded with the GB model enabling to additionally correct for hydrogen bonding, the resulting model is called GBSA. The parameters for the GBSA model are currently available at https://github.com/grimme-lab/gbsa-parameters and can be read in with ParamFile and will setup the complete GeneralisedBorn{} input.

Note that the GB(SA) model implemented is only available for finite systems.

ParamFile	s			
Solvent	m	not has ParamFile		
FreeEnergyShift	r			
Temperature	r		298.15 K	
State	s		gsolv	
Kernel	s		Still	
BornScale	r			
BornOffset	r			
OBCCorrection	3r		1.00, 0.80, 4.85	
CM5	m			71
Radii	m		vanDerWaalsRadiiD3	
Descreening	m			
Cutoff	r		35 AA	
SASA	m			73
HBondCorr	l	has SASA		
HBondStrength	m	HBondCorr = Yes		
ALPB	l		No	
Alpha	r	ALPB = Yes	0.571412	

ParamFile Reads in a parameter file for GBSA, specifying this keyword automatically provides the Solvent information, and defaults values for FreeEnergyShift, BornOffset, BornScale, SASA{} Descreening, HBondCorr and HBondStrength. Usually no other keywords need to be specified when ParamFile is present.

This file is additionally searched for in the directory specified with the DFTBPLUS_PARAM_DIR environment variable.

Solvent Descriptors of the solvent, can be load from a database by providing the solvent name as string in the FromName{} method or by specifying the necessary values with the FromConstants{} method. FromConstants{} requires the dielectric constant as real for Epsilon, the molecular mass in MolecularMass [*mass*] and the solvent density in Density [*massdensity*]. MolecularMass and Density only affect the calculation if “reference” is chosen as state of the solution.

Available solvent names and associated constants are given in Appendix I.

FreeEnergyShift [*energy*] Shift for free energy calculations.

Temperature Temperature for free energy calculations. Default is ambient temperature: 298.15

K. Only affects the calculation for if “reference” or “mol1bar” is chosen as state of the solution.

State [*energy*] Reference state of the solution for free energy calculations. The calculated state shift is added to the free energy shift. Takes “gsolv” (default), “reference” or “mol1bar”. The reference state “gsolv” corresponds to 1 l of ideal gas and 1 l of liquid solution, “reference” corresponds to 1 bar of ideal gas and 1 mol/L of liquid solution at infinite dilution, “mol1bar” corresponds to 1 bar ideal gas and 1 mol/L of liquid solution.

Kernel Interaction kernel for the screened Coulomb operator. Possible options are the canonical “Still” kernel[86] and the “P16” kernel.[51]

BornScale Value for scaling of Born radii.

BornOffset [*length*] Offset value for Born radii calculation.

OBCCorrection Parameters for Onufriev–Bashfold–Case volume polynomial to correct Born radii calculation. The default values 1.0, 0.8, 4.85 correspond to GB$^{\mathrm{OBC}}$II, alternatively 0.8, 0.0, 2.91 can be used for GB$^{\mathrm{OBC}}$I.[71]

CM5 Use the charge model 5 to correct the atomic partial charges before evaluating the Born energy.

Radii Atomic radii for each element in Bohr, either takes VanDerWaalsRadiiD3{} for DFT-D3 van-der-Waals radii (can be overwritten) or requires to provide Values{} for all species. Both methods accept [*length*] units.

Descreening Descreening values for each species. Disabled by Unity{} method or enabled by providing Values{} for each species.

Cutoff [*length*] Real space cutoff for the calculation of the Born radii.

HBondCorr Include an empirical hydrogen bond correction. Only available for GBSA models.

HBondStrength Hydrogen bonding strength for each species used in the empirical hydrogen bond correction. To disable the correction for species not involved in hydrogen bonding, set the value to zero.

ALPB Use analytical lineared Poisson Boltzmann (ALPB) model[85] instead of Generalised Born. The main difference is a molecular shape dependent contribution for charged systems.

Alpha Alpha parameter in the ALPB model, should be kept to 0.571412 which was derived from first principles.[85]

Example for a GB model with CS_2 as solvent:

```
Hamiltonian = DFTB {
  Solvation = GeneralisedBorn { # GFN2-xTB/GBSA(CS2)
    Solvent = fromName { "cs2" }
    FreeEnergyShift [kcal/mol] = 2.81072250
    Radii = vanDerWaalsRadiiD3 [Bohr] {}
    Descreening = Values {
       H = 0.93699367
       C = 0.83307834
```

```
      N = 1.02661619
      O = 0.96508008
    }
    BornScale = 1.40636177
    BornOffset [Bohr] = 1.653719965215E-03
    OBCCorrection = {1.00 0.80 4.85}
    Cutoff = 40
  }
}
```

Example for a GBSA model for water

```
Hamiltonian = DFTB {
  Solvation = GeneralisedBorn { # GFN2-xTB/GBSA(water)
    Solvent = fromConstants {
      Epsilon = 80.2
      MolecularMass [amu] = 18.0
      Density [kg/l] = 1.0
    }
    FreeEnergyShift [kcal/mol] = 1.16556316
    BornScale = 1.55243817
    BornOffset = 2.462811043694508E-02
    Radii = vanDerWaalsRadiiD3 {}
    Descreening = Values {
      H = 0.71893869
      C = 0.74298311
      N = 0.90261230
      O = 0.75369019
    }
    SASA {
      ProbeRadius = 1.843075777670416
      Radii = vanDerWaalsRadiiD3 {}
      SurfaceTension = Values {
        H = -3.34983060E-01
        C = -7.47690650E-01
        N = -2.31291292E+00
        O =  9.17979110E-01
      }
    }
    HBondCorr = Yes
    HBondStrength = Values {
        H = -7.172800544988973E-02
        C = -2.548469535762511E-03
        N = -1.976849501504001E-02
        O = -8.462476828587280E-03
    }
  }
}
```

Charge Model 5

The charge model 5 (CM5)[60] can be used to correct partial charges by

$$q_{\mathrm{A}}^{\mathrm{CM5}} = q_{\mathrm{A}} + \sum_{B} D_{\mathrm{A\text{–}B}} \exp[-\alpha(R_{\mathrm{AB}} - R_{\mathrm{A}}^{\mathrm{cov}} - R_{\mathrm{B}}^{\mathrm{cov}}] \tag{2.5}$$

The pairwise parameters $D_{\mathrm{A\text{–}B}}$ are fixed to the original published ones, while the exponent α and the covalent radii $R_{\mathrm{A}}^{\mathrm{cov}}$ default to the published parameters but can be adjusted in the input.

alpha	r	2.474 1/AA
Radii	m	atomicRadii{}
Cutoff	r	30.0

alpha [*1/length*] Exponent of the CM5 correction.

Radii Atomic covalent radii for each species, either takes AtomicRadii{} for default atomic radii[58] (can be overwritten) or requires to provide Values{} for all species. Both methods accept [*length*] units.

Cutoff [*length*] Real space cutoff for the calculation of the CM5 correction.

```
CM5 {
  Alpha = 1.30918451402600
  Radii = AtomicRad {}
}
```

Conductor like screening model

The conductor like screening model (COSMO)[43] allows to include dielectric screening effects in quantum chemical calculations. DFTB+ uses the fast domain decomposition algorithm[56] to minimise the overhead of solving the COSMO equations in the SCC iterations.

In the domain decomposition algorithm the solvation energy is given by

$$E_{\mathrm{solv}} = \frac{1}{2}\left(\frac{1}{\varepsilon_{\mathrm{in}}} - \frac{1}{\varepsilon_{\mathrm{out}}}\right) \sum_{j=1}^{N_{\mathrm{atoms}}} \sum_{l=0}^{N_{\mathrm{basis}}} \sum_{m=-l}^{l} [\Psi_j]_l^m \, [X_j]_l^m \tag{2.6}$$

where N_{basis} is the maximum degree of the expansion in spherical harmonics, $[\Psi_j]_l^m$ describes the interaction of the tight-binding Hamiltonian with the spherical harmonics and $[X_j]_l^m$ are the local solution of the block sparse COSMO equation

$$LX = g \tag{2.7}$$

The right-hand-side of the system of equations depends on the solute potential at the cavity by

$$[g_j]_l^m = \sum_{n=1}^{N_{\mathrm{grid}}} w_n Y_l^m(\vec{y}_n) U_n^j \Psi_n^j \tag{2.8}$$

where Ψ_n^j is the solute's potential at the n-th grid point.

FreeEnergyShift	r	0.0 Eh	
Temperature	r	298.15 K	
State	s	gsolv	
AngularGrid	i		
Radii	m	vanDerWaalsRadiiD3	
RadiiScaling	r		
Solver	m	DomainDecomposition	
SASA	m		73

Solvent Descriptors of the solvent, can be load from a database by providing the solvent name as string in the FromName{} method or by specifying the necessary values with the FromConstants{} method. FromConstants{} requires the dielectric constant as real for Epsilon, the molecular mass in MolecularMass [*mass*] and the solvent density in Density [*massdensity*]. MolecularMass and Density only affect the calculation if "reference" is chosen as state of the solution. Setting Epsilon to Inf results in the ideal conductor limit.

Available solvent names and associated constants are given in Appendix I.

FreeEnergyShift [*energy*] Shift for free energy calculations.

Temperature Temperature for free energy calculations. Default is ambient temperature: 298.15 K. Only affects the calculation for if "reference" or "mol1bar" is chosen as state of the solution.

State [*energy*] Reference state of the solution for free energy calculations. The calculated state shift is added to the free energy shift. Takes "gsolv" (default), "reference" or "mol1bar". The reference state "gsolv" corresponds to 1 l of ideal gas and 1 l of liquid solution, "reference" corresponds to 1 bar of ideal gas and 1 mol/L of liquid solution at infinite dilution, "mol1bar" corresponds to 1 bar ideal gas and 1 mol/L of liquid solution.

AngularGrid Size of the angular Lebedev–Laikov integration grid.[53] The grid size mainly determines the computational cost of solving the COSMO equations, too small grid sizes can lead to significant errors due to missing rotational invariance. Possible values are 6, 14, 26, 38, 50, 74, 86, 110, 146, 170, 194, 230, 266, 302, 350, 434, 590, 770, 974, 1202, 1454, 1730, 2030, 2354, 2702, 3074, 3470, 3890, 4334, 4802, 5294, 5810.

Radii Atomic radii for each element, either takes VanDerWaalsRadiiD3{} for DFT-D3 van-der-Waals radii (can be overwritten) or requires to provide Values{} for all species. Both methods accept [*length*] units. Alternatively, VanDerWaalsRadiiCosmo{} and VanDerWaalsRadiiBondi{} can be used to provide van-der-Waals radii. Note that Bondi radii[59] are only available for main group elements and Cosmo optimised radii default to 2.223 AA if no optimised values are available.

RadiiScaling Allows to scale the provided radii by the provided value. Values greater one between 1.3 and 1.5 are recommended when using DFT-D3 van-der-Waals radii, while for Bondi radii usually values between 1.2 and 1.3 are used. Cosmo optimised radius should be used with a scaling factor of unity.

Solver for the COSMO equation. Currently only DomainDecomposition{} is implemented (default). If specified explicitely, following parameters can be set:

MaxMoment Maximum moment for spherical harmonic expansion in the ddCOSMO equations. Moments of 6 and higher are recommended.

Regularisation / Regularization Regularisation factor for the ddCOSMO equation. A value of 0.2 is recommended

Accuracy Convergence criteria for the ddCOSMO equations, this value effects the cost of the solver in the self-consistent charge iterations. The convergence thresholds are automatically decreased for the gradient calculations to avoid numerical noise.

Example for a COSMO model

```
Hamiltonian = Dftb {
  Solvation = Cosmo {
    Solvent = FromConstants {
      Epsilon = 80.2
      MolecularMass [amu] = 18.0
      Density [kg/l] = 1.0
    }
    FreeEnergyShift [kcal/mol] = 0.0
    Radii = VanDerWaalsRadiiD3 {}
    RadiiScaling = 1.55
    AngularGrid = 110
    Solver = DomainDecomposition {
      MaxMoment = 10
      Accuracy = 1e-8
      Regularisation = 0.2
    }
  }
}
```

Solvent area model

The non-polar solvation free energy can be estimated from the solvent accessible surface area (SASA) by

$$\Delta G_{\text{non-polar}} = \sum_{\text{A}} \gamma_{\text{A}} \sigma_{\text{A}} \tag{2.9}$$

where γ_{A} is the surface tension and σ_{A} is the accessible surface area of each atom. To calculate the latter, the molecule is assumed as a convolution of spheres which is probed by a probe sphere rolled around the surface. Here a smooth numerical integration approach is employed.[40]

To use the non-polar surface area model in an calculation use the SASA{} method in the input to Solvation. This model is currently only available for finite systems.

ProbeRadius	r		
Smoothing	r		0.3 AA
Offset	r		2.0 AA
Tolerance	r		1.0e-6
AngularGrid	i		230
Radii	m		vanDerWaalsRadiiD3
SurfaceTension	m		

ProbeRadius [*length*] Radius of the probe sphere used to determine the accessible surface area.

Smoothing [*length*] Smoothing parameter for numerical integration.

Offset [*length*] This offset value is added on the realspace cutoff radius for the neighbourlist generation. The realspace cutoff is determined automatically from the probe radius, the largest atomic radius and the smoothing parameter.

Tolerance Minimal value of surface area contribution of a grid point to be accounted for as SASA.

AngularGrid Size of the angular Lebedev–Laikov integration grid.[53] The grid size mainly determines the computational cost of evaluating the accessible surface area, too small grid sizes can lead to significant errors due to missing rotational invariance. A safe choice should be 230 grid points per atom. Possible values are 6, 14, 26, 38, 50, 74, 86, 110, 146, 170, 194, *230*, 266, 302, 350, 434, 590, 770, 974, 1202, 1454, 1730, 2030, 2354, 2702, 3074, 3470, 3890, 4334, 4802, 5294, 5810.

Radii Atomic radii for each element, either takes VanDerWaalsRadiiD3{} for DFT-D3 van-der-Waals radii (can be overwritten) or requires to provide Values{} for all species. Both methods accept [*length*] units.

SurfaceTension Surface tension parameter for each species in dyn/cm.

```
Hamiltonian = DFTB {
  Solvation = SASA { # GFN1-xTB/GBSA(Toluene)
    ProbeRadius [AA] = 1.59772343
    Smoothing [AA] = 0.3
    Offset [AA] = 2
    AngularGrid = 230
    Radii = vanDerWaalsRadiiD3 {}
    SurfaceTension = Values {
      H = -1.52312760
      C = -2.92375089
      O = 0.79482640
    }
  }
}
```

External electric fields and dipole moments

For externally applied homogeneous electric fields (see section 2.4.13) where the solvent modifies the dielectric surroundings, DFTB+ can approximate the local field corrections as due to a spherical cavity surrounding the solvated system. The applied field in the external medium is then rescaled by

$$\vec{E}_{eff.} = \vec{E}_{ext.} \frac{3\varepsilon_r}{2\varepsilon_r + 1}$$

Likewise the dipole moment of the system is scaled to give the corresponding field in the external medium:

$$\vec{\mu}_{ext.} = \vec{\mu} \frac{3\varepsilon_r}{2\varepsilon_r + 1}$$

The rescaling can be turned off with the following keyword:

RescaleSolvatedFields	l		Yes

RescaleSolvatedFields Enables or disables field scaling for applied homogeneous fields and dipole moments.

2.4.18 Halogen corrections

The HalogenXCorr keyword includes the halogen correction of Ref. [50]. This is fitted for the DFTB3-D3 model and the 3ob-3-1 parameter set. The correction is only relevant for systems including interactions between {O,N}–{Cl,Br,I} pairs of atoms.

2.4.19 Hydrogen corrections

There are currently two available methods to correct hydrogen interactions (mainly hydrogen bonds) in the HCorrection environment:

Damping

The Damping method modifies the short range contribution to the SCC interaction between atoms A and B with the damping factor

$$e^{-\left(\frac{U_{Al}+U_{Bl}}{2}\right)^{\zeta} r_{AB}^2}$$

provided that at least one of the two atoms is hydrogen [27, 93]. (U_{Al} and U_{Bl} are the Hubbard U values of the two atoms for the l-shell, r_{AB} is the distance between the atoms.) An atom is considered to be a hydrogen-like atom, if its mass (stored in the appropriate homonuclear SK-file) is less than 3.5 amu. The Exponent keyword in this environment sets the parameter ζ for the short range damping:

```
HCorrection = Damping {
  Exponent = 4.05
}
```

Table 2 of reference [27] gives suggested values of the exponent for different DFTB2 and DFTB3 models applied to light atoms bonded to hydrogen.

DFTB3-D3H5

DFTB3-D3H5 [80] is a variant of DFTB3 with additional corrections for non-covalent interactions (dispersion and hydrogen bonds). It consists of a third-order DFTB calculation using the 3OB parameter set, but where the gamma-function damping (Damping method above) is replaced by the H5 correction and an additional D3 dispersion correction in included. This method also includes a repulsive term which is added to prevent unphysically close approach of pairs of hydrogen atoms [81].

Setting the HCorrection environment to H5{} activates this correction for hydrogen bonds [80]. If no additional parameters are provided in the input, suitable values for H-{O,N,S} systems are used (the correction was developed for the DFTB3/3OB model and parameters).

```
HCorrection = H5 {}
```

Note: It was found that DFTB3 overestimates the strength of H-bonds involving the terminal nitrogen of an azide group, and the published results in Ref. [80] were obtained with the H5 correction switched off for these specific atoms. To reproduce this behavior in a system containing nitrogen in several environments, a new atom type with a different name but the same DFTB parameters can be used for specific N atoms to which the correction should not be applied.

If you want to specify the parameters manually, H5 accepts following options, corresponding to terms in Ref. [80]:

RScaling	r	0.714
WScaling	r	0.25
H5Scaling	m	

RScaling Global scaling factor, s_r, when calculating the position of the correcting gaussian functions:

$$r_0 = s_r \left(r_{\text{vdW}}(X) + r_{\text{vdW}}(H) \right).$$

WScaling Global scaling factor, s_W, when calculating the width of the correcting gaussian functions. The full-width at at half-maximum of the gaussian, w, is normalised to be 1 for a unit value of WScaling:

$$w = \frac{s_w \left(r_{\text{vdW}}(X) + r_{\text{vdW}}(H) \right)}{2\sqrt{2 \ln 2}}.$$

H5Scaling Atom type specific scaling pre-factor, $k_{X\text{H}}$, of the correcting gaussian functions when calculating the SCC-interaction:

$$\gamma_{X\text{H}}^{\text{H5}} = \gamma_{X\text{H}} \left(1 + k_{X\text{H}} \exp \left(-\frac{(r_{X\text{H}} - r_0)^2}{2w^2} \right) \right).$$

You will have to specify one value for each of the chemical species you would like to correct (see the example below). Explicitly setting a negative value (e.g. -1.0) for a given atom type switches off the correction for hydrogen bonds involving that type of atom. In the special cases of N, O or S, if you do not specify a value (and do not disable the contribution by using -1.0), the default value from the reference paper will be used [80]. For any other omitted atom types, the code defaults to a choice of -1.0 (no correction).

```
Hamiltonian = DFTB {
  :
  HCorrection = H5 {
    RScaling = 0.714
    WScaling = 0.25
    H5Scaling {
      O = 0.06
      N = 0.18
      S = 0.21
    }
  }
  :
}
```

Note: The van der Waals radii (r_{vdW}) of atoms are also required. DFTB+ stores these for most of the periodic table, but for cases that are not available their contribution to this correction are neglected.

For a DFTB3-D3H5 calculation, a specific parametrisation of the D3 dispersion has to be used. In addition to setting up appropriate values of the D3 parameters, as discussed in Ref. [80], the hydrogen–hydrogen repulsion of Ref. [81] has to also be activated. The complete input is:

```
Hamiltonian = DFTB {
  :
  Dispersion = DftD3 {
    Damping = ZeroDamping {
      sr6 = 1.25
      alpha6 = 29.61
    }
    s6 = 1.0
    s8 = 0.49
    HHRepulsion = Yes
  }
  :
}
```

2.4.20 RangeSeparated

The RangeSeparated keyword specifies the use of a range separated hybrid functional. Currently, only the long-range corrected hybrid functional (LC) [66, 57] is implemented. There, the electrostatic interaction is split up into long and short ranged components according to

$$\frac{1}{r} = \frac{1-e^{-\omega r}}{r} + \frac{e^{-\omega r}}{r},$$

with the range-separation parameter ω, which is set in the Slater-Koster files. The option should only be used with corresponding parameter sets created for use with long-range corrections.

The RangeSeparated keyword expects either None (default – no use of range-separated hybrid functional) or the LC{ } block as value. The latter has the following option:

Screening	m	Thresholded {}

Screening Choice of the screening method. The following choices are possible:

Thresholded {} Screening according to estimated magnitude of terms. This is faster than the NeighbourBased method below, but does not support all of the cases (restarting and spin polarisation).

Threshold	r	1e-6
CutoffReduction	r	0.0

Threshold Threshold, below which elements are considered to be zero.

CutoffReduction [*length*] Reduces the spatial cutoff, beyond which the overlap between atoms is considered to be zero. This can be used as an additional tweak to speed up the LC-calculation, but make sure first, that your results do not change considerably. Default: 0.0 – no reduction, using the cutoff from the SK-files.

NeighbourBased Uses a purely neighbour-list based algorithm. This algorithm is usually considerably slower than the Thresholded.

CutoffReduction	r	0.0

CutoffReduction [*length*] See description in the Thresholded block.

MatrixBased Uses a matrix-matrix multiplication based algorithm. This can be faster than other two algorithms.

Example for thresholded screening with customised threshold value.

```
RangeSeparated = LC {
  Screening = Thresholded {
    Threshold = 1e-5
  }
}
```

Example for neighbour list based screening with customised cutoff reduction:

```
RangeSeparated = LC {
  Screening = NeighbourBased {
    CutoffReduction [AA] = 2.0
  }
}
```

Example for matrix-matrix multiplication based method:

```
RangeSeparated = LC {
  Screening = MatrixBased {}
}
```

2.4.21 On site corrections

This block enables corrections for on-site matrix elements which improve the description of multi-centre integrals [26] leading to, for example, improved hydrogen-bond energies [20].

For each chemical species, the spin-same-spin and spin-different-spin constants should be specified for all combinations of atomic shells. **note:** the matrix of constants is symmetric and the purely s-with-s entries are zero (the code ignores their value due to symmetry).

Example:

```
OnSiteCorrection= {
    # same spin oxygen
    Ouu = {0.00000  0.08672
           0.08672 -0.00523}
    # hetero-spin oxygen
    Oud = {0.00000  0.14969
           0.14969  0.03834}
    # H all zero
    Huu = {0}
    Hud = {0}
}
```

Some on-site constants are given in appendix J.

2.4.22 Differentiation

Calculations of forces currently require the numerical derivatives of the overlap and non-self-consistent Hamiltonian. This environment controls how these derivatives are evaluated.

Note: In earlier DFTB+ versions (up to version 1.2), differentiation was done using finite difference derivatives with a step size of 0.01 atomic units. If you want to reproduce old results, choose the FiniteDiff method and set the step size explicitly to this value.

FiniteDiff{}

Finite difference derivatives with a specified step size

Delta	r	epsilon$^{1/4}$

Delta [*length*] Step size

Richardson{}

Extrapolation of finite difference via Richardson's deferred approach to the limit (in principle the most accurate of the currently available choices).

2.4.23 ForceEvaluation

Chooses the method for evaluating the electronic contribution to the forces.

'traditional' Uses the "traditional" DFTB-force expression, given for example, in Ref. [23].

'dynamics' Force expression from Ref. [7]. This choice should be used if forces are being calculated with non-converged charges (e.g. when doing XLBOMD dynamics). **Note:** this force expression is only compatible with the Fermi filling (see keyword Filling, p. 47.)

'dynamicsT0' Simplified dynamic force expression valid for electronic temperature $T = 0$ K [7]. This choice should be used if forces are calculated with non-converged charges and the electronic temperature is zero (e.g. when doing XLBOMD dynamics at $T = 0$ K).

Note: that XLBOMD calculations (Section 2.3.8) are not able to use the 'traditional' forces.

Example:

```
ForceEvaluation = 'dynamics'
```

2.5 Options

This block collects some global options for the run.

WriteAutotestTag	l		No
WriteDetailedXML	l		No
WriteResultsTag	l		No
WriteDetailedOut	l		Yes
RestartFrequency	i	Driver = {}, SCC = Yes	20
RandomSeed	i		0
MinimiseMemoryUsage	l		No
TimingVerbosity	i		0
ShowFoldedCoords	l	Periodic = Yes	No
WriteHS	l		No
WriteRealHS	l		No
ReadChargesAsText	l	ReadInitialCharges = Yes	No
WriteCharges	l		Yes
WriteChargesAsText	l		No
SkipChargeTest	l	ReadInitialCharges = Yes	No
BinaryAccessTypes	s | 2s		"stream"

WriteAutotestTag Turns the creation of the autotest.tag file on and off. (This file can get quite big and is only needed for the autotesting framework.)

WriteDetailedXML Turns the creation of the detailed.xml file on and off. (The detailed.xml file is needed among others by the waveplot utility for visualising molecular orbitals.)

WriteResultsTag Turns the creation of the results.tag file on and off. (That file is used by several utilities processing the results of DFTB+.) For a description of the file format see p. 104.

WriteDetailedOut Controls the creation of the file detailed.out (see p. 103). Since this contains the detailed information about the last step of your run, you shouldn't turn it off without good reasons.

RestartFrequency Specifies the interval at which charge restart information should be written to disc for static SCC calculations. Setting it to 0 prevents the storage of restart information. If running an MD calculation, see also section 2.3.8 regarding MDRestartFrequency.

RandomSeed Sets the seed for the random number generator. The value 0 causes random initialisation. (This value can be used to reproduce earlier MD calculations by setting the initial seed to the same value.)

MinimiseMemoryUsage / MinimizeMemoryUsage Tries to minimise memory usage by storing various matrices on disc instead of keeping them in memory. Set it to Yes to reduce the memory requirement for calculations with many k-points or spin polarisation. Note: Currently this option has no effect and you will get a warning if setting it to be Yes.

TimingVerbosity Level of information regarding CPU and wall clock timings of sections of the code, higher values becoming more verbose. Setting this parameter to 0 or below suppresses any information being printed (default). Setting it to -1 includes all measured timings.

ShowFoldedCoords Print coordinates folded back into the central cell, so if an atom moves outside the central cell it will reappear on the opposite side. The default behaviour is to use unfolded coordinates in the output. (Please note, that this option only influences how the coordinates are printed and written, it does not change the way, periodic systems are treated internally.)

WriteHS Instructs the program to build the square Hamiltonian and overlap matrices and write them to files. The output files are hamsqrN.dat and oversqr.dat, where N enumerates the spin channels. For a detailed description of the file format see p. 105.

Note: If either of the options WriteHS or WriteRealHS are set to Yes, the program only builds the matrices, writes them to disc and then stops immediately. No diagonalisation, no SCC-cycles or geometry optimisation steps are carried out. You can use the ReadInitialCharges option to build the Hamiltonian with a previously converged charge distribution.

WriteRealHS Instructs the program to build the real space (sparse) Hamiltonian and overlap matrices and write them to files. The output files are hamreal.dat and overreal.dat. For a detailed description of the file format see p. 105.

Note: If either of the options WriteHS or WriteRealHS are set to Yes, the program only builds the matrices, writes them to disc and then stops immediately. No diagonalisation, no SCC-cycles or geometry optimisation steps are carried out. You can use the ReadInitialCharges option to build the Hamiltonian with a previously converged charge distribution.

ReadChargesAsText If No, the program expects the file charges.bin to contain starting charges stored in binary. If Yes, then charges.dat should contain a text file of this data. See section 3.7.

WriteCharges Turns the creation of the charges.bin (or charges.dat) file on and off. (This might e.g. prove to be useful to avoid unnecessary I/O load during geometry optimization or MD, if a later restart of the calculation is not needed.)

WriteChargesAsText If No, the program stores charges in the binary file charges.bin, while if Yes then charges.dat contains text of this data. See section 3.7.

SkipChargeTest If Yes, testing of whether the charges read from file match the total charge (and magnetisation) specified in the DFTB+ input (if relevant) is performed. Skipping this test (setting to No) may be useful if restarting from a charges generated for a similar system with slightly different total charge or magnetisation. Similarly, in the event of serious instabilities in the SCC cycle, the generated charge restart file may fall outside of the check-sum tolerances, hence this option allows a re-start. Finally, in the case of user edited charges.dat file (see section 3.7), the check-sum this option removed the requirement that the checksum values in the file match the charges.

BinaryAccessTypes Specifies the default access format for binary (non-human-readable) files. It contains either one string specifying the same type for both reading and writing those files, or two strings separately specifying the types for reading and writing. Each value must be either "stream" or "sequential". The default is "stream" for both actions.

Starting with version 23.1, DFTB+ consistently uses stream-I/O for binary files in order to produce files (e.g. eigenvec.bin, charges.bin, etc.), which can be processed by programs written in other languages (Python, C). In earlier versions most binary files were written using sequential I/O, which were not transferable between languages or even between Fortran compilers. You can read in or write out files in this old format by setting the appropriate action to "sequential".

Example:

```
# Use old sequential I/O for input and output of binary files
BinaryAccessTypes = "sequential"
```

```
# Use the old format when reading and the new format when writing binary files
BinaryAccessTypes = "sequential" "stream"
```

2.6 Analysis

This block collects some options to analyse the results of the calculation and/or calculate properties.

AtomResolvedEnergies	l		No	
MullikenAnalysis	l		Yes	
WriteCosmoFile	l		No	
CM5	m	MullikenAnalysis = Yes		71
WriteNetCharges	l	MullikenAnalysis = Yes	No	
ProjectStates	m		{}	
Localise	m		{}	
WriteEigenvectors	l		No	
EigenvectorsAsText	l	WriteEigenvectors = Yes	No	
WriteBandOut	l		Yes	
Printforces	l		No	
ElectrostaticPotential	m	SCC = Yes	{}	84
Polarisability	m	Hamiltonian = DFTB	{}	86
ResponseKernel	m	Hamiltonian = DFTB	{}	86

AtomResolvedEnergies Specifies whether the contribution of the individual atoms to the total energies should be displayed or not.

MullikenAnalysis If Yes, the results of a Mulliken analysis of the system is given.

WriteNetCharges If Yes, the net charges (component of the charge associated only with the on-site part of the atomic orbitals, not the off-site hybridisation).

WriteCosmoFile If Yes, the cavitity information is written as a cosmo file to dftbp.cosmo. Only works if the solvation model is COSMO.

CM5 If present the charge model 5 (CM5)[60] corrected atomic partial charges will be written.

WriteEigenvectors Specifies, if eigenvectors should be printed in eigenvec.bin. For a description of the file format see p. 106.

EigenvectorsAsText If eigenvectors are being written, specifies if a text version of the data should be printed in eigenvec.out. For a description of the file format see p. 106.

WriteBandOut Controls the creation of the file band.out which contains the band structure in a more or less human friendly format.

Printforces If Yes, forces are reported, even if not needed for the actual calculation (e.g. static geometry calculation).

2.6.1 ProjectStates

ProjectStates evaluates the Mulliken projection of electronic states onto specific regions of the system being modelled (partial density of states – PDOS). The format of the projected data files is similar to band.out, but the second column is the fraction of the state within that region, instead of its occupation number (for non-collinear and spin-orbit calculations, three additional columns for the magnetisation of the state are also given).

Each region for projection is specified within a Region{} block, with the following options

Atoms	(i|s)+	-
ShellResolved	l	No
OrbitalResolved	l	No
Label	s	"region*i*"

ShellResolved Project onto separate atomic shells of the region. These are taken in order of increasing shell number of the atoms. ShellResolved = Yes is only allowed, if all the selected atoms are of the same type.

OrbitalResolved Project onto separate atomic orbitals of the region. These are taken in order of increasing shell number of the atoms. As with ShellResolved, this only allowed, if all the selected atoms are of the same type.

Atoms Specification of the atoms over which to make the projection. The atoms can be specified via index selection expressions, as described in appendix B.7.

Label Prefix of the label for the resulting file of data for this region. The default is "region*i*.out" where *i* is the number of the region in the input. In the case that ShellResolved = Yes, the shell index is appended, so that files with names "Label.*j*.out" are written. For OrbitalResolved = Yes, the shell and then *m*-value is appended, so that files with names "Label.*j*.*m*.out" are written.

Example:

```
ProjectStates = {
  Region = {           # first region
    Atoms = 23:25 27     # atoms 23, 24, 25 and 27
  }
  Region = {
    Atoms = N            # All nitrogen atoms
    ShellResolved = Yes   # s and p shells separated instead of atomic PDOS
    Label = "N"          # files N.1.out and N.2.out for s and p states
  }
}
```

2.6.2 Localise / Localize

Convert the single particle states of the calculation to localised orbitals via a unitary transformation. Localised orbitals span the same states as the occupied orbitals, so are equivalent to the usual valence band states, but are more localised in space. Currently only PipekMezey localisation is supported (but not for non-collinear or spin-orbit calculations).

Pipek-Mezey [76] localisation transforms the occupied orbitals such that the square of the Mulliken charges for each orbital is maximised. The resulting localised states are output as localOrbs.out and localOrbs.bin following the format given in appendix 3.6 for eigenvec.out and eigenvec.bin.

Tolerance	r		1E-4
MaxIterations	i		100

Tolerance Cut off for rotations in the localisation process.

MaxIterations Maximum number of total sweeps to perform.

For systems with non-gamma-point *k*-points, no further options are available.

```
Analysis = {
  Localise = {
    PipekMezey = {
      # These are the default options, which are also set if the bracket is left empty.
      Tolerance = 1.0E-4
      MaxIterations = 100
    }
  }
}
```

For molecular and gamma point periodic calculations there are two implementations available, Dense = Yes will use the $O(n^4)$ scaling conventional algorithm, while Dense = No, uses the default sparse method which *may* have better scaling properties.

Dense	l		No
SparseTolerances	r+	Dense = No	1E-1 1E-2 1E-6 1E-12

Dense Selects the conventional method (Yes) using Jacobi sweeps over all orbital pairs or (No) uses the default sparse method.

SparseTolerances The sparse method introduces support regions during evaluation to increase performance, and these requires a set of tolerances to determine the regions to be used (these are listed in decreasing order, i.e., with tighter tolerances as the localisation proceeds).

2.6.3 ElectrostaticPotential

Evaluates the electrostatic potential at specified points in space for SCC calculations. This data is accumulated in a specified text file.

OutputFile	s		"ESP.dat"
AppendFile	l	MD or geometry optimisation	No
Softening	r		1E-6
Points	(3r)+	Grid not set	{}
Grid	m	Points not set	{}

OutputFile Text file to store the potential. If external electric fields are present, an additional column gives their values. See p. 107 for a description of the file.

AppendFile If running calculations with multiple geometries, should the OutputFile be appended or only contain the last potential information?

Softening [*length*] Modifies the plotted potential to remove the $r = 0$ divergence of $1/r$, by setting ε and instead plotting $1/\sqrt{r^2+\varepsilon^2}$. Internal potential calculations are unaffected, only the exported data.

Points [*length*] List of cartesian points at which to evaluate the electrostatic field. In the case Periodic = Yes, the modifier "F" may instead be used to specify the points as fractions of the lattice vectors.

Grid [*length*] Specification of a regular 1, 2 or 3 dimensional grid of points. In the case Periodic = Yes, the modifier "F" may instead be used to specify the points as fractions of the lattice vectors.

GridPoints	3i		
Origin	3r		
Spacing	3r		
Directions	9r	Modifier not F	1 0 0 0 1 0 0 0 1

Spacing Separation between points in each direction. This inherits the modifier for Grid.

Origin Location of first point in the grid. This inherits the modifier for Grid.

GridPoints Number of points in each of the three direction of the grid (a value of 1 places all points at the Origin of that direction).

Directions Set of 3 cartesian vectors along which the grid will become aligned. This can rotate, skew, *etc.* the grid. The vectors are internally normalised, but must be independent.

2.6.4 Response properties by perturbation

For cases where the ground state eigenvalues are available, the output of the eigenvalue derivatives are described in are section 3.1.1.

There are common options for the perturbations, which appear outside their respective blocks in the Analysis environment:

ConvergedPerturb	l	SCC = Yes	Yes
MaxPerturbIter	i	SCC = Yes	100
PerturbDegenTol	r		128
PerturbEta	r	Dynamic Polarisation	1E-8
PerturbSccTol	r	SCC = Yes	1e-5

ConvergedPerturb If true, requires that the perturbation self-consistently converges. If false, the code can continue with a warning, but will return NaN for properties evaluated at that stage.

MaxPerturbIter Maximal number of self-consistent cycles to reach convergence for perturbation evaluation of properties. If convergence is not reached after the specified number of steps, the program stops. This behaviour can be overridden using ConvergedPerturb, but will then produce Not a Number (NaN) for non-convergent properties. **Note:** For calculations using *k*-points, the default number of self-consistent iterations will default to 1 in cases where a band structure is being plotted (see KLines{} in section 2.4.11).

PerturbDegenTol In the case of degenerate systems, determines the number of times machine epsilon that separated eigenvalues are considered non-degenerate.

PerturbEta In the case of finite frequency perturbations, a small imaginary constant, of magnitude PerturbEta is used in the expression to avoid divergences at resonant conditions.

PerturbSccTol Stopping criteria for the perturbation self-consistency. Specifies the tolerance for the maximum difference in any charge derivatives between two cycles.

Polarisability

Evaluates electric polarisability using coupled-perturbed linear response (hence can only be used with cases initially evaluating unperturbed eigenvalues/band structure). Currently only the DFTB hamiltonian is supported (see section 2.4.1, REKS and Delta-DFTB calculations are also not currently supported).

In the case of degenerate single particle levels, the reported level derivatives (see p. 103) for those states are ordered in increasing value.

Static	1		Yes
Frequencies	r+		–
FrequencyRange	3r		–

Static Evaluates polarisability with respect to cartesian directions of a static uniform electric field.

Frequencies [*Frequency*] Evaluates polarisability at a list of finite frequencies ($\{\omega\}$).

FrequencyRange [*Frequency*] Specifies equispaced driving frequencies for polarisabilities, listed as *starting ending step-size* frequencies (for example if calculating a spectrum).

Examples Static polarisability, with an adjusted degeneracy tolerance:

```
Analysis {
  Polarisability = {} # static enabled by default
  PerturbDegenTol = 1024 # levels within 1024 x machine tolerance
}
```

Dynamic polarisability at specified frequencies

```
Analysis {
  Polarisability = {
     Static = No # static field response turned off
     FrequencyRange [eV] = 1.0 4.0 0.1 # 1 to 4 eV with steps of 0.1
  }
}
```

ResponseKernel

Evaluates responses to potentials added at individual atomic sites, using coupled-perturbed linear response (hence can only be used with cases initially evaluating unperturbed eigenvalues/band structure). Currently only the DFTB hamiltonian is supported (see section 2.4.1, REKS and Delta-DFTB calculations are also not currently supported).

The same options as Polarisability are supported, but in addition:

RPA	1	SCC = Yes	No

RPA For self-consistent calculations, enables RPA evaluation, instead of self-consistent calculation of the response. This is equivalent to evaluating response without relaxation of the system.

Examples Static polarisability for an Scc = Yes calculation.

```
Analysis {
  ResponseKernel = {
    # static enabled by default
    RPA = Yes # non-self consistent perturbation
  }
}
```

Dynamic polarisability at a specified frequency

```
Analysis {
  Polarisability = {
     Static = No # static field response turned off
     Frequencies [eV] = 1.0 4.0 # evaluate at 1 and 4 eV
  }
}
```

2.7 ExcitedState

This block collects some options to calculate excited states.

Casida	p	SCC = Yes	{}
PP-RPA	p	SCC = Yes	{}

2.7.1 Casida

This tag contains the specifications for a time-dependent DFTB calculation, based on linear response theory [67].

Note: the DFTB+ binary must be compiled with linear response calculations enabled to make use of these features (the ARPACK [55] library or ARPACK-ng [1] is required).

The calculation of vertical excitation energies and the corresponding oscillator strengths as well as excited state geometry optimisation can be performed with these options, details of the resulting output files are given in appendix 3.10. Linear response theory is currently implemented only for the SCC-DFTB level of theory and molecular systems.[6] Excitations can be calculated for fractional occupations and collinear spin-polarisation. Forces (and hence geometry optimisation or MD) are available for collinear spin-polarised systems but not for fractional occupations. The specifications for this block have the following properties:

Note: Excited state calculations with the RangeSeparated functionals (Section 2.4.20) can be performed for molecular (non-periodic) systems that do not have fractional fillings. The various window options (see below) are not yet available for range separated calculations.

[6]Excitation energies can also be calculated for gamma point periodic systems, but will be incorrect for delocalised excitations or for charge transfer-type excited states.

NrOfExcitations	i		-	
StateOfInterest	i		0	
Symmetry	s	SpinPolarisation = {}	-	
Diagonaliser	m		{}	89
EnergyWindow	r		FORTRAN HUGE()	
OscillatorWindow	r		-1	
WriteTransitions	l		No	
WriteTransitionCharges	l	RangeSeparated and StateOfInterest non-zero	No	
WriteSPTransitions	l		No	
WriteMulliken	l		No	
WriteCoefficients	l		No	
WriteEigenvectors	l		No	
WriteDensityMatrix	l		No	
TotalStateCoeffs	l	WriteCoefficients = Yes	No	
WriteXplusY	l		No	
WriteTransitionDipole	l		No	
ExcitedStateForces	l	Printforces = Yes	Yes	
CacheCharges	l		Yes	
StateCouplings	2i		-	
OptimiserCI	m	StateCouplings non-zero	{}	90

NrOfExcitations Specifies the number of vertical excitation energies to be computed for every symmetry (singlet or triplet). It is recommended that a value slightly greater than the actual number of the states of interest is specified (the eigenvalue solver may not converge to the right roots otherwise).

StateOfInterest Specifies the target excited state or states that should be calculated. These are numbered from the first (lowest) excited state as 1, and so on. If the absorption spectrum at a given geometry is required (i.e., a single-point calculation), this parameter should be set to zero (default) and the Driver section (2.3) should be left empty (forces will not be available). A value less than 0 requests that the state with the largest dipole transition moment be found (again a single-point calculation). In the output Total Energy and Total Mermin free energy refer to StateOfInterest, while the extrapolated energy and force related energy refer to the ground state.

Symmetry Specifies the spin symmetry of the excited states being computed: “singlet”, “triplet” or “both”. This tag is only applicable for spin restricted calculation. For calculations in the “triplet” or “both” cases, SpinConstants must be supplied (see p. 43).

EnergyWindow [*energy*] Energy range above the last transition at NrOfExcitations to be included in excited state spectrum calculation.

OscillatorWindow [*Dipole moment*] Screening cut-off below which single particle transitions are neglected in excitation spectra calculations. This selects from states above the top of the EnergyWindow (if present). This keyword should not be used if calculating forces or other excited state properties.

WriteTransitions If set to Yes, the file TRA.DAT is created. This file contains a description of each requested excited state in terms of its single-particle transitions.

WriteSPTransitions If set to Yes, the file SPX.DAT is created, which contains the spectrum at the uncoupled DFTB level (i.e. the single-particle excitations).

WriteTransitionCharges For range separated calculations (p. 77) setting this to Yes generates the file ATQ.DAT containing the Mulliken charges for a specific transition (chosen by StateOfInterest).

WriteMulliken If set to Yes, the files XCH.DAT and XREST.DAT are created. The former contains atom-resolved Mulliken (gross) charges for the excited state of interest, the latter the excited-state dipole moment of the state.

WriteCoefficients If set to Yes, the file COEF.DAT is created. This file contains the complex eigenvectors (molecular orbital coefficient) for the excited state of interest. They are derived from the relaxed excited state density matrix.

WriteEigenvectors If set to Yes, the file excitedOrbs.bin is created (and excitedOrbs.out depending on whether EigenvectorsAsText is true). This file contains the natural orbitals for the specified excited state.

WriteDensityMatrix If set to Yes, the file(s) DM?.bin are created. These hold the excited state density matrices, in the atomic orbital basis, for the evaluated states.

TotalStateCoeffs Option to control data from WriteCoefficients or WriteEigenvectors. If set to No the total charge density of the output orbitals corresponds to the change in charge from the ground to excited state. If set to Yes instead it corresponds to the total charge density in the excited state.

WriteXplusY If set to Yes, the file XplusY.DAT is created. This file contains the RPA vector $(X+Y)_{ia}^{I\Sigma}$ for all excited states (c.f., Eqn. (18) in Ref. [36]).

WriteTransitionDipole If set to Yes, the file TDP.DAT is created. This file contains the Mulliken transition dipole for each excited state.

ExcitedStateForces If set to Yes, evaluated forces include the contributions from an excited state of interest. By default, it is set to Yes if forces are being calculated (for example in geometry optimisation) and to No otherwise. By setting it explicitly to No, you can calculate the excitations during a molecular dynamics simulation that is being driven by the ground state forces only.

CacheCharges If set to No, transition charges are calculated on the fly during the excited states calculation, instead of being cached. This makes the calculation considerably slower, but can help to decrease memory use substantially, if you are short on memory.

StateCouplings Evaluates non-adiabatic coupling vectors between specified many-body states of your system. This keyword requires two values, m and n, with $m < n$ and state 0 being defined as the ground state. The couplings are computed according to the method described in [68, 69].
Example: Couplings = {0 2} would generate the the couplings between states $0 \to 1$, $0 \to 2$ and $1 \to 2$. The results are written to the file NACV.DAT (See p. 110). Couplings are usually calculated for single point geometries, as the file NACV.DAT is overwritten along a trajectory.

Diagonaliser / Diagonalizer

Specifies which iterative diagonaliser should be used to solve the Casida equations. The keyword expects either a Arpack{} or Stratmann{} block.

Arpack	p	RangeSeparated = No
Stratmann	p	

Arpack{} refers to the Implicitly Restarted Arnoldi Method [55] and is available only for the conventional TD-DFTB method with local functionals (RangeSeparated=None). It has two options:

WriteStatusArnoldi	l	No
TestArnoldi	l	No

WriteStatusArnoldi If set to Yes, the file ARPACK.DAT is created, which allows the user to follow the progress of the Arnoldi diagonalisation.

TestArnoldi If set to Yes, the file TEST_ARPACK.DAT is created, which gives data on the quality of the resulting eigenstates.

The Stratmann{} diagonaliser [88] (see also [49, 19]) is available for local and range separated functionals and is the recommended choice for small systems and/or a small number of excited states to solve for. It has one option:

SubSpaceFactor	i	20

SubSpaceFactor The initial subspace for the diagonalisation is given by SubSpaceFactor times the number of excited states to solve for. Small values of SubSpaceFactor speed up the calculation, but might lead to convergence problems. In this case the calculation needs to be restarted with an increased value of SubSpaceFactor.

OptimizerCI

Invokes search for conical intersections (CI) between the two states indicated by the StateCouplings keyword. Note that the two states must be direct neighbours (i.e., $n = m + 1$). There is currently one optimizer available.

The Bearpark{} optimizer (see [11]) minimizes a gradient that is based on the non-adiabatic coupling vector and the difference gradient of the two potential energy surfaces in question. The parameters given in the GeometryOptimisation block control the optimization as usual. Since the CI optimization is numerically challenging, we recommend the SteepestDescent driver. The energy gap between the two states in question is written as entry Energy gap CI to the output.

Bearpark{} has one option, which is relevant for the CI between the ground and first excited state:

EnergyShift	r	0.0

EnergyShift Shift of the excited state PES during optimization. Should be brought to zero on restarting with a better optimised structure.

As discussed by Harabuchi et al. [34], EnergyShift helps to avoid SCC convergence issues close to a ground-to-excited CI. It is recommended to first approach the CI with a larger value of EnergyShift, stop the optimization before the loss of self-consistency occurs, and then restart the optimization with successively smaller values of EnergyShift. Note that specifying StateOfInterest leads only to computation of the corresponding energy, but not the force.

2.7.2 PP-RPA

This tag contains the specifications for the calculation of excitation energies using the particle-particle random phase approximation (pp-RPA) [92]. This approach, unlike time-dependent DFTB, allows the computation of double and charge-transfer transitions. However it has the limitation that the computed excitations have to involve, at least partially, the highest occupied molecular orbital (HOMO) of the system.

For the computation of the excitation energies of a neutral N-electron system, one needs to set up a ground state calculation for the two-electron deficient $(N-2)$ system, i.e. a net `Charge` = +2.0 calculation. Please note that if `Charge` is set to 0.0 (the default value), the obtained transition energies will correspond to a net negative -2 charged system (i.e. $N+2$ electrons). The system of interest must be closed-shell, therefore the calculation must also be spin-restricted and performed for an even number of electrons.

The pp-RPA method is currently implemented only for SCC-DFTB level excitations, but can be performed on top of both SCC-DFTB or range-separated reference ground state calculations. The SCC-DFTB calculations can be performed for molecular or gamma point periodic systems, but the range-separated calculations can only use molecular boundary conditions.

The specifications for this block have the following properties:

NrOfExcitations	i	-
Symmetry	s	-
NrOfVirtualStates	i	0
TammDancoff	l	No
HHubbard	p	-

NrOfExcitations Specifies the number of vertical excitation energies to be computed for each symmetry (singlet or triplet).

Symmetry Specifies the spin symmetry of the excited states being computed: `singlet`, `triplet` or `both`. Please note that, `triplet` and `both` are effectively similar, as both singlet and triplet excitation energies will be printed out if either of these keywords are used.

NrOfVirtualStates Optional orbital constraint to speed up the calculation. It specifies the number of virtual states entering the pp-RPA equation. If set to zero or greater than the total number of virtual states of the system no constraint will be applied.

TammDancoff If set to Yes, the Tamm-Dancoff approximation will be employed. This will speed up the calculation.

HHubbard Hubbard-like parameters for each atom type including only the Hartree kernel. Values of some of these parameters are given in appendix K.

The output of the pp-RPA calculation are described in section 3.12.

2.8 ElectronDynamics

Calculate the real-time propagation of both the electronic state of the system and the nuclei within the Ehfenfest approximation in the presence of an optional external time-dependent perturbation, as described in Ref. [14]. The real-time propagation is available with DFTB as well as xTB Hamiltonians.

Steps	i		-	
TimeStep	r		-	
FieldStrength	r		-	
Perturbation	m		None	94
EnvelopeShape	m	Perturbation = Laser{}	Constant	95
Populations	l		No	
Restart	l		No	
RestartFromAscii	l	Restart = Yes	No	
WriteRestart	l		Yes	
WriteAsciiRestart	l	WriteRestart = Yes	No	
WriteFrequency	i		50	
WriteEnergyAndCharges	l			95
RestartFrequency	i		Steps/10	
Forces	l		No	
IonDynamics	l		No	
InitialTemperature	r	IonDynamics = Yes	-	
MovedAtoms	(ils)+	IonDynamics = Yes	1:-1	
Velocities	(3r)*	IonDynamics = Yes	-	
Pump	l		No	
PumpProbeFrames	i	Pump = Yes	-	
ProbeProbeRange	2r	Pump = Yes	0 Steps*TimeStep	
Probe	l		No	
EulerFrequency	i		0	
WriteBondEnergy	l		No	
WriteBondPopulation	l		No	
WriteAtomicEnergies	l		No	
FillingsFromFile	l		No	

Steps Number of propagation steps to perform.

TimeStep [*time*] Time interval between two electronic propagation steps.

FieldStrength [*Electric field strength*] Peak intensity of the applied time-dependent electric field.

Populations If time-dependent (ground state) molecular orbital populations should be calculated. This is done by projecting the electronic state onto the ground state molecular orbitals. If Yes, they are saved every WriteFrequency steps to molpopul1.dat (and molpopul2.dat is the calculation is collinearly spin polarised).

Restart If calculation should be restarted. A restart file tddump.bin (or tddump.dat) must be available in the working directory.

RestartFromAscii If this is a restart calculation (Restart = Yes), read tddump.dat instead of tddump.bin in the working directory.

WriteRestart If a restart tddump file should be written. Details of the contents of the file are discussed in appendix L.0.3.

WriteAsciiRestart If a restart file is being written, select between producing tddump.bin (No) or if set to Yes, tddump.dat.

RestartFrequency Number of steps every which the system's state should be written to restart file.

WriteFrequency Number of steps every which the atomic charges and molecular orbital populations (if Populations = Yes) are written to file.

Forces If forces on all atoms should be calculated and written to forcesvst.dat.

IonDynamics If nuclei should be propagated in the dynamics using the Velocity Verlet algorithm. If set to Yes, then either the InitialTemperature keyworkd or the Velocities keyword must be set. The positions and velocities every WriteFrequency steps will be saved in XYZ format to tdcoords.xyz.

MovedAtoms List of atoms that will be moved during the dynamics. The atoms can be specified via index selection expressions, as described in appendix B.7.

InitialTemperature [*energy*] Create starting velocities for the Ehrenfest MD according to the Maxwell-Boltzmann distribution at the specified temperature. This keyword is redundant in the case of specified initial velocities or for a restarted trajectory, so is omitted in those cases.

Velocities [*velocity*] Specified atomic velocities for all the atoms of the given structure (including "velocities" for any stationary atoms, which are silently ignored). (**Note:** if the velocities from a previous (MD or Ehrenfest) run are used, the velocities printed in the XYZ files are specified in Å/ps, so this should be set in the input). See section 2.3.8.

Pump If this trajectory corresponds to the dynamics under a pump pulse, with the intention to probe the system afterwards at different delay times, to simulate a pump-probe transient absorption experiment. The effect of the keyword is to write dump files *i*ppdump.bin ($i = 0,\ldots,$PumpProbeFrames) inside the pump_frames directory, containing the state of the system (density matrix, coordinates, velocities) every a given number of steps.

PumpProbeFrames Number of total snapshots of the system in the pump trajectory that will be saved for future probe simulations. These are spaced uniformly in time over the PumpProbeRange.

PumpProbeRange [*time*] The time range (initial time and final time, separated by a space) between which the snapshots will be dumped to file.

Probe If this is the simulation of a probe. If set to yes, then automatically Restart = Yes, WriteRestart = No, and only the dipole moment output files will be written, since this keyword is used together with a Kick perturbation to probe the system. Notice that the dump file that contains the state of the system that you want to be probed must be present and renamed to tddump.bin (or tddump.dat if using the ascii version of this data).

EulerFrequency Number of steps every which an Euler integration step is done in the electronic propagation (that normally uses the Leapfrog algorithm). The default value ensures no Euler steps are done during the dynamics. If used, must be set to something larger than 50.

WriteBondEnergy If the pairwise non-self-consistent part of the bond energy, $BE(t)$, should be written to the bondenergy.bin file every WriteFrequency steps. The populations are evaluated as

$$BE_{AB}(t) = \sum_{\mu\in A}\sum_{\nu\in B} \rho_{\mu\nu}(t) H^0_{\mu\nu}(t),$$

where A and B are the atoms in the central cell and S is the overlap matrix. They are stored in binary format (see 3.11.9).

WriteBondPopulation If the pairwise time-dependent bond Mulliken-like population, $BP(t)$, should be written to the bondpop.bin file every WriteFrequency steps. The populations are evaluated as

$$BP_{AB}(t) = \sum_{\mu \in A} \sum_{\nu \in B} \rho_{\mu\nu}(t) S_{\mu\nu}(t),$$

where A and B are the atoms in the central cell and S is the overlap matrix. They are stored in binary format (see 3.11.10).

WriteAtomicEnergies If the atom-resolved contributions to the total energy shpuld be written to the atomenergies.dat file.

FillingsFromFile If the initial fillings (molecular orbital occupations) should be read from a file named “fillings.ini” present in the working directory. If used, the ground state fillings are replaced by these and used to build the initial density matrix.

2.8.1 Perturbation{}

Determines the type of perturbation that is applied.

None{} No time-dependent perturbation is included, free dynamics are calculated.

Kick{} Perform Dirac-delta perturbation (or a *kick*) to the density matrix.

PolarisationDirection	s		-
SpinType	s	SpinPolarisation = Colinear {}	Singlet

PolarisationDirection / PolarizationDirection The cartesian axis for the kick: "x", "y" or "z". If set to "all", calculates the three directions x, y, z consecutively. For polarisation direction x,y,z the dipole moment output file will be named mux.dat, muy.dat, muz.dat, respectively.

SpinType Must be either “Singlet” or “Triplet” for singlet or triplet spectra, respectively. Only implemented for collinear spin polarisation.

Laser{} Apply time-dependent sinusoidal perturbation to the system.

PolarisationDirection	3r	-
ImagPolarisationDirection	3r	0 0 0
LaserEnergy	r	
Phase	r	0
ExcitedAtoms	(ils)+	1:-1

PolarisationDirection / PolarizationDirection Vector along which the electric field is polarised.

ImagPolarisationDirection / ImagPolarizationDirection Imaginary part of the polarisation vector. Useful for circularly polarised fields.

LaserEnergy [*energy*] Energy $\hbar\omega$ of the laser.

Phase [*Angular units*] Optional initial phase of laser field (inradians by default), such that sinusoidal component of the field is described by $\sin(\omega t + \phi)$.

ExcitedAtoms List of atoms that will be excited by the laser. The atoms can be specified via index selection expressions, as described in appendix B.7.

Note: when working with periodic systems, **the polarisation direction must be orthogonal to the periodic directions** of the system. Therefore, the code does not work with 3D periodic systems with laser perturbations.

KickAndLaser{} Apply a Kick plus a Laser to the system. Useful for probing the excited state on the system while being driven by a laser. The keywords are a combination of the ones for kick and laser.

KickPolDir	i	-
LaserPolDir	3r	-
LaserImagPolDir	3r	0 0 0
LaserEnergy	r	-
LaserStrength	r	-
Phase	r	0
ExcitedAtoms	(ils)+	1:-1

KickPolDir Same as PolarisationDirection in Kick.

SpinType Same as Kick.

LaserPolDir Same as PolarisationDirection in Laser.

LaserImagPolDir Same as ImagPolarisationDirection in Laser.

LaserEnergy [*energy*] Same as in Laser.

LaserStrength Peak intensity of the applied laser (in this mode, FieldStrength is the kick intensity).

Phase Same as in Laser.

ExcitedAtoms Same as in Laser.

2.8.2 WriteEnergyAndCharges

This enables the write out of additional data during the propagation. By default, excitations using Kick or KickAndLaser excitations sets this to false, while Laser or None generates the extra files. This option controls write out of Mulliken charges during the calculation (qvst.dat) and energy components (energyvst.dat). If forces are evaluated this keyword also enables writing of (forcesvst.dat) or if ion dynamics are allowed the atom coordinates during the calculation (tdcoords.xyz).

2.8.3 EnvelopeShape{}

Determines the envelope $f(t)$ of the laser, such that the laser field is $E(t) = E_0 f(t) \sin(\omega t + \phi)$

Constant{} Constant envelope and equal to $f(t) = 1$. Produces continuous wave laser.

Gaussian{} Applies a Gaussian envelope function $f(t) = \exp\big(-(t-t_m)^2/\beta^2\big)$, where t_m is the time at which the pulse is centered and $\beta = \tau/2\sqrt{\pi}$, τ being the duration of the pulse.

Time0	r	0
Time1	r	-

Time0 Time at which the pulse starts.

Time1 Time at which the pulse ends. Is equal to Time0 + τ.

Sin2{} Applies a $\sin^2$ envelope function:

$$f(t) = \begin{cases} \sin^2(\pi(t-t_0)/\tau) & t_0 \le t \le t_0+\tau \\ 0 & t < t_0 \text{ or } t > t_0+\tau. \end{cases}$$

The properties of this method are the same as for the Gaussian (Time0, Time1) and have the same meaning.

Example:

```
ElectronDynamics = {
  Steps = 40000
  TimeStep = 0.1
  # Total time will be then 4000 a.u. = 96.8 fs
  FieldStrength [v/a] = 0.01
  Perturbation = Laser {
    PolarisationDirection = 0.5 0.5 0
    LaserEnergy [ev] = 2.55
  }
  EnvelopeShape = Sin2 {
    Time1 [fs] = 30.0
  }
  Populations = Yes
  IonDynamics = Yes
  InitialTemperature [k] = 0.0
  Pump = Yes
  PumpProbeFrames = 1000
  PumpProbeRange [fs] = 0.0 50.0
  EulerFrequency = 200
}
```

2.9 REKS

This block collects some options to calculate REKS in the context of DFTB. The Reks keyword expects either None (default – no use of REKS calculation) or the SSR22{ } block as value.

SSR22	p	SCC = Yes, SpinPolarisation = {}, SpinConstants $\neq$ {}	None

2.9.1 SSR22

This tag contains the specifications for a DFTB/SSR(2,2) calculation [54], based on ensemble DFT theory.

Note: the DFTB+ binary can be compiled with OpenMP (not MPI) parallelisation and DFTB/SSR calculation is not compatible with time-dependent DFTB calculation. In addition, it is not compatible with spin-polarisation, but it requires spin constants to treat open-shell microstates.

In general, REKS calculation can be classified as single-state REKS, SA-REKS and SI-SA-REKS. In single-state REKS, only ground state is calculated and it can treat the state with multireference character. SA-REKS and SI-SA-REKS can calculate the vertical excitation energies. The difference is that the state-interaction term is considered in SI-SA-REKS so that more accurate states can be generated. The corresponding oscillator strengths as well as excited state geometry optimisation can be performed with these options, details of the resulting output files are given in appendix 3.13.

In the context of DFTB, current REKS calculation is compatible with following functionalities. The range-separated functional, external point charges and dispersion corrections can be calculated with single-state REKS, SA-REKS or SI-SA-REKS. For the periodic system, only gamma point sampling is supported with REKS. Especially, the stress evaluation and lattice optimisation is possible with only single-state REKS. The specifications for this block have the following properties:

Energy	m		{}
TargetState	i		1
TargetMicrostate	i	Functional ≠ { "PPS" }, StateInteractions = No	0
ReadEigenvectors	l		No
FonMaxIter	i		20
Shift	r		0.3
SpinTuning	(r)*		{}
TransitionDipole	l	Functional ≠ { "PPS" }, TargetMicrostate = 0	No
Gradient	m		ConjugateGradient {}
RelaxedDensity	l		No
NonAdiabaticCoupling	l	Functional ≠ { "PPS" }, StateInteractions = Yes	No
VerbosityLevel	i		1

Energy Choice of energy evaluation in REKS method. This Energy block has following options:

Functional	(s)*	
IncludeAllStates	l	No
StateInteractions	l	No

Functional Specifies the minimised energy functional in DFTB/SSR. This keyword reads a block consisted of the energy functionals that you want to include in the calculation. In DFTB/SSR(2,2), there are two possible choices for the minimzied energy functionals. One is PPS and the other is (PPS+OSS)/2. The former represents single-state REKS and the latter shows SA-REKS or SI-SA-REKS. The detailed form of the block is shown in below examples. The inclusion of state-interaction terms is determined by StateInteractions.

IncludeAllStates If set to Yes, all computable energy states from current energy functionals are included for SA-REKS or SI-SA-REKS calculations. When you calculate single-state REKS, this option does not affect the result of calculation. The PPS and OSS states are calculated when this option sets to No, while the additional DES state can be included if this sets to Yes. If you want to the doubly-excited configuration, please set to Yes. The detailed explanation about PPS, OSS and DES states is given in the Ref. [54]

StateInteractions If set to Yes, the state-interaction terms between SA-REKS states is included, thus it generates SI-SA-REKS states. In general, SI-SA-REKS state can provide more reliable state when you want to compute the excited states.

TargetState Specifies the target state that should be calculated. These are numbered from the ground state as 1, and so on. Note that the ordering of this option is different with the option StateOfInterest in time-dependent DFTB calculation.

TargetMicrostate Specifies the target microstate that should be calculated. The electronic configuration is given in the Ref. [54] or the source code of DFTB+. In SSR(2,2), fifth microstate is triplet configuration, thus this microstate can be roughly considered as triplet state.

ReadEigenvectors If set to Yes, the initial molecular orbitals are read from the eigenvec.bin file. If not, the initial orbitals are obtained from the diagonalisation of non-SCC Hamiltonian.

FonMaxIter Specifies the maximum number of iterations used in the optimisation of fractional occupation numbers. In general, the value of 20 is enough to converge the fractional occupation numbers in SCC cycle.

Shift Specifies the level shift value used in SCC cycle. The shift value should be increased to converge the SCC cycle when the orbital energies of the active orbitals are close to each other.

SpinTuning Specifies the scaling constants for atomic spin constants. DFTB/SSR sometimes shows wrong spin contribution for triplet microstate, thus the scaling of atomic spin constants are needed to generate correct spin contribution for each microstate. The standard to determine the scaling constants is provided in the Ref. [54]. The number of elements of SpinTuning block becomes the number of atomic species, and the ordering of the elements is same as the ordering of atomic species in input geometry file.

TransitionDipole If set to Yes, the file tdp.dat is created. This file contains a description of transition dipole moment between the electronic states in SA-REKS or SI-SA-REKS.

Gradient Choice of gradient solver used in CP-REKS equations. This Gradient block has the following choices:

ConjugateGradient Uses a congugate-gradient based algorithm. This algorithm is usually recommended since it is considerably faster than other algorithms.

CGmaxIter	i	20
Tolerance	r	1e-8
Preconditioner	l	No
SaveMemory	l	No

CGmaxIter Specifies the maximum number of iterations used in the conjugate-gradient based algorithm. In general, the value of 20 is enough to solve the CP-REKS equations.

Tolerance Specifies the tolerance used in the conjugate-gradient based algorithm.

Preconditioner If set to Yes, it uses a preconditiner in the conjugate-gradient based algorithm. In general, the convergence speed is increased when this option sets to Yes, thus this option is recommended.

SaveMemory If set to Yes, some variables (an orbital hessian matrix and the H-xc kernel) which need large memory allocation are saved in the memory. If these variables are saved, then the computational speed also increases but it shows large memory allocation, increasing as $O(N_{\text{basis}}^4)$. If set to No, the CP-REKS equations are solved without saving these variables, thus it is relatively slower than the case that you set to Yes. In general, No option is recommended for large systems.

Direct Uses a direct matrix-inversion multiplication algorithm. This algorithm is usually considerably slow.

RelaxedDensity If set to Yes, the file relaxed_charge.dat is created. This file contains a description of relaxed charges for TargetState or TargetMicrostate. The relaxed charges can be used with external point charges in QM/MM calculations.

NonAdiabaticCoupling If set to Yes, the nonadiabatic couplings between SI-SA-REKS states are calculated. This option cannot be used in single-state REKS or SA-REKS state.

VerbosityLevel Specifies the printing level in standard output. This option determines the output up to energy information (VerbosityLevel = 0), gradient information (VerbosityLevel = 1), detailed information about SCC cycle and timing in gradient calculation (VerbosityLevel = 2).

Example for 3state SI-SA-REKS calculation with nonadiabatic couplings and modified spin constants:

```
Reks = SSR22 {
  Energy = {
    Functional = { "PPS" "OSS" }
    IncludeAllStates = Yes
    StateInteractions = Yes
  }
  TargetState = 2
  TargetMicrostate = 0
  ReadEigenvectors = No
  FonMaxIter = 30
  Shift = 0.3
  SpinTuning = { 3.0 3.0 }
  Gradient = ConjugateGradient {
    CGmaxIter = 100
    Tolerance = 1.0E-8
    Preconditioner = Yes
    SaveMemory = Yes
  }
  RelaxedDensity = Yes
  NonAdiabaticCoupling = Yes
  VerbosityLevel = 1
}
```

2.10 ParserOptions

This block contains the options, which are effecting only the behaviour of the HSD parser and are not passed to the main program.

ParserVersion	i	current input version
WriteHSDInput	l	Yes
IgnoreUnprocessedNodes	l	No
StopAfterParsing	l	No

ParserVersion Version number of the input parser, which the input file was written for. If you are using an input file, which was created for an older version of DFTB+, you should set it to the parser version number of that code version. (The parser version number is printed at the beginning of the program run to the standard output.) DFTB+ internally converts the input to its current format. The processed input (written to dftb_pin.hsd) is always in the current format, and the ParserVersion property in it is always set to be the current parser version.

The parser version will be inferred in case InputVersion is set in the level above, in this case ParserVersion cannot be specified.

WriteHSDInput Specifies, if the processed input should be written out in HSD format. (You shouldn't turn it off without really good reasons.)

IgnoreUnprocessedNodes By default the code stops if it detects unused or erroneous keywords in the input, which probably indicates error(s) in the input. This *dangerous* flag suspends these checks. Use only for debugging purposes.

StopAfterParsing If set to Yes, the parser stops after processing the input and written out the processed input to the disc. It can be used to make sanity checks on the input without starting an actual calculation.

2.11 Parallel

This block contains the options, which are effecting the parallel behaviour of the code. They only take effect, if the code was compiled with MPI-support.

Groups	i	1
UseOmpThreads	l	.false.
Blacs	p	{}

Groups Number of process groups. Specifying more than one process group enables parallelisation over k-points and spin, as processes in different process groups are working on different k-points and spins at the same time. The number of process groups must be a divisor of the total number of MPI-processes. Default: 1 (all processes work at the same k-point and spin at a given time). Note that transport calculations between contacts are currently incompatible with multiple process groups (see section 4).

UseOmpThreads Enables the usage of OpenMP-threads (hybrid MPI/OpenMP-parallelisation). In order to prevent you from accidentally running more processes and threads than appropriate for your hardware, this feature is turned off by default. Consequently in this case the MPI-parallelised binary will stop if the maximal number of OpenMP-threads is greater than one when DFTB+ is started. (You can usually set the number of maximally allowed OpenMP-threads by setting the OMP_NUM_THREADS environment variable in your shell.)

You can enable this option if you wish to run DFTB+ with hybrid parallelisation. You would then typically start fewer MPI-processes than physical cores on each node and also set the number of threads accordingly. This is currently an experimental feature in DFTB+ and is recommended for experienced users only.

Blacs Contain BLACS specific settings. Currently only supports BlockSize, which specifies the row and column block size for the block-cyclic distributions (with default size of 32).

Example:

```
Parallel {
  Groups = 2
  Blacs {
    BlockSize = 64
  }
}
```

Chapter 3

Output of DFTB+

This chapter contains the description of some of the output files of DFTB+ where the output format is not self documenting. Unless indicated otherwise, numbers in the output files are given in atomic units (with Hartree as the energy unit).

3.1 band.out

This contains the band energies and occupation of levels in electron volts and electron charge units as columns one and two. The file is printed if WriteBandOut = Yes (see section 2.6). Blocks of numerical results start with a line which labels the k-point and spin channel for the energies.

See the DP_TOOLS package for utilities for converting the data in this file into band-structures and density of states information suitable for plotting.

3.1.1 dE_band.out

If perturbation calculations of electric field polarizability (p. 86) is enabled and WriteBandOut = Yes, this file contains the derivatives of the band energies in electron volts / a.u. for each field direction, k-point and spin channel.

3.2 detailed.out

This file contains details of the total energy and its components, as well as optional information on forces, atomic charges and other properties. It is intended for quick viewing, while values given to more significant figures are available in results.tag.

Some of the information available in the file will also depend on the method being used in the calculation. For example, not all electronic solvers make the ground state electronic entropy available, hence only the internal energy would be quoted. Similarly, while the free energy of the system which when differentiated by atomic coordinates or boundary conditions gives the forces or stresses (printed as *Force related energy*) this is not currently available for some types of non-equilibrium transport calculations.

Some of the common energy results printed in this file are:

TS	Product of the electron entropy and temperature
Total Electronic energy	The non-SCC energy plus other contributions to the electronic energy (SCC, spin, ...)
Repulsive energy	The pairwise contribution to the total energy
Total energy	Sum of electronic energy
Extrapolated to 0	Estimated zero temperature energy if at finite temperatures
Total Mermin free energy	$U-TS$, relevant free energy at finite temperatures
Force related energy	Free energy relevant to forces in the system
Gibbs free energy	Energy corrected by $-pV$, i.e. the pressure and volume
MD Kinetic Energy	Kinetic energy of atoms in molecular dynamics
Total MD Energy	Sum of finite temperature electronic, repulsive and atomic kinetic energies

Where available the Fermi level μ (i.e. the chemical potential of the electrons in the system) is also printed. For systems with an externally fixed Fermi level (i.e. where the total charge can change), this contribution is included in the Force related energy:

$$\Delta E = +q_{\text{total}}\mu,$$

but for calculations with fixed numbers of electrons it is not included in this energy. **Note:** The total energy reference may not match your required case in some situations, for example a shift with respect to the average electrostatic potential (in periodic cases) or whether the chemical potential should be with respect to the valence band maximum may be needed (see for example the discussion in Ref. [52]).

3.3 results.tag

This contains machine readable results labeled with the type and size of the data blocks. The results are given in atomic units and are formatted as:

```
label            :type:shape:
```

The variable type is real, complex, integer or logical. The shape information is
:ndim: size_1,size_2,...,size_{ndim}:
where ndim is the number of dimensions, organised with the Fortran convention and of size $\text{size}_1 \times\text{size}_2 \times\text{size}_2 \times \ldots$.

In the special case of scalar variables the shape is :0:.

A typical example of mixed scalar and both one and two dimensional results would be similar to:

```
mermin_energy       :real:0:
 -0.672967201447815E+000
total_energy        :real:0:
 -0.672879398682698E+000
forces              :real:2:3,3
 -0.243590222274811E+000 -0.199780753617099E-001 -0.000000000000000E+000
  0.465478448963764E+000 -0.228550455811745E+000 -0.000000000000000E+000
 -0.221888226688953E+000  0.248528531173455E+000 -0.000000000000000E+000
gross_atomic_charges:real:1:3
  0.171448741143825E+000 -0.254714832621691E+000  0.832660914778645E-001
```

3.4 hamsqrN.dat, oversqr.dat

The files hamsqrN.dat and oversqr.dat contain the square (folded) Hamiltonian and overlap matrices. The number N in the filename hamrealN.dat indicates the spin channel. For spin unpolarised calculation it is 1, for spin polarised calculation it is 1 and 2 for spin-up and spin-down, respectively while for non-collinear spin it is charge, x, y and z for 1, 2, 3 and 4. Spin orbit is not currently supported for this option.

Only non-comment lines (lines not starting with "#") are documented:

- Flag for signalling if matrix is real (REAL), number of orbitals in the system (NALLORB), number of kpoints (NKPOINT). For non-periodic (cluster) calculations, the number of kpoints is set to 1.
- For every k-point:
  - Number of the k-point. For molecular (non-periodic) calculations only 1 k-point is printed.
  - The folded matrix for the given k-point. It consists of NALLORB lines $\times$ NALLORB columns. If the matrix is not complex (REAL is F), every column contains two numbers (real and imaginary part).

Each column / row of a square dense matrix belongs to an (atom, orbital) index pair, with the orbital index running faster. The order of the atoms follows their order in the input geometry. The orbitals are ordered by increasing magnetic quantum number (m) of each atomic angular shell (l), using real tesserals functions. The angular shells are ordered according to the input to the calculation. In most cases this leads to orbitals ordered by increasing (l, m), but for non-minimal or reordered basis functions the shell order (l) may not necessarily be in an increasing order. For the usual minimal basis, the standard orbital order runs as

$$s,\, p_y,\, p_z,\, p_x,\, d_{xy},\, d_{yz},\, d_{z^2},\, d_{xz},\, d_{x^2-y^2},\, f_{y(3x^2-y^2)},\, f_{xyz},\, f_{yz^2},\, f_{z^3},\, f_{xz^2},\, f_{z(x^2-y^2)},\, f_{x(x^2-3y^2)},$$

for an atom, where only the orbitals present in the selected basis of the atom are considered. In other more general cases, while m is always in increasing order for a given shell, l may be reordered, have multiple shells, or be lacking particular l values.

The files are produced if requested by WriteHS = Yes (see section 2.5).

3.5 hamrealN.dat, overreal.dat

The files hamrealN.dat and overreal.dat contain the real space Hamiltonian and overlap matrices. The number N in the filename hamrealN.dat indicates the spin channel. For spin unpolarised calculation it is 1, for spin polarised calculation it is 1 and 2 for spin-up and spin-down, respectively, while for non-collinear spin it is charge, x, y and z for 1, 2, 3 and 4. Spin orbit is not currently supported for this option.

Note: The sparse format contains only the "lower triangle" of the real space matrix. For more details about the format and how to obtain the upper triangle elements, see reference [6]. Also note, that for periodic systems the sparse format is based on the *folded* coordinates of the atoms, resulting in translation vectors (ICELL) which look surprising at first glance.

Only non-comment lines (lines not starting with "#") are documented:

- Number of atoms in the system (NATOM)
- For every atom:
  - Atom number (IATOM), number of neighbours including the atom itself (NNEIGH), number of orbitals on the atom (NORB)
- For every neighbour of every atom:
  - Atom number (IATOM1), neighbour number (INEIGH), corresponding image atom to the neighbour in the central cell (IATOM2F), coefficients of the translation vector between the neighbour and its corresponding image (ICELL(1), ICELL(2), ICELL(3)). Between the coordinates of the neighbour $\mathbf{r}_{\text{INEIGH}}$ and the image atom $\mathbf{r}_{\text{IATOM2F}}$ the relation

    $$\mathbf{r}_{\text{INEIGH}} = \mathbf{r}_{\text{IATOM2F}} + \sum_{i=1}^{3} \text{ICELL}(i)\,\mathbf{a}_i$$

    holds, where $\mathbf{a}_i$ are the lattice vectors of the supercell.
  - The corresponding part of the sparse matrix. The data block consists of NORB(IAT1) lines and NORB(IAT2F) columns.

The files are produced if requested by WriteRealHS = Yes (see section 2.5).

3.6 eigenvec.out, eigenvec.bin

These files contain the eigenvectors from the Hamiltonian, stored either as plain text (eigenvec.out) or in the native binary format of your system (eigenvec.bin).

The plain text format file eigenvec.out contains a list of the values of the components of each eigenvector for the basis functions of each atom. The atom number in the geometry, its chemical type and the particular basis function are listed, followed by the relevant value from the current eigenvector and then the Mulliken population for that basis function for that level. The particular eigenvector, k-point and spin channel are listed at the start of each set of eigenvector data. In the case of non-collinear spin, the format is generalised for spinor wavefunctions. Complex coefficients for both the up and down parts of the spinors are given (instead of single eigenvector coefficient) followed by four values – total charge, then (x, y, z) magnetisation.

The binary format file eigenvec.bin contains the (unique) runId of the DFTB+ simulation which produced the output followed by the values of the eigenvectors. The eigenvector data is ordered so that the individual components of the current eigenvector are stored, with subsequent eigenvectors for that *k*-point following sequentially. All *k*-points for the current spin channel are printed in this order, followed by the data for a second channel if spin polarised.

The files are produced if requested by setting WriteEigenvectors = Yes, with EigenvectorsAsText being also required to produce the plain text file (see section 2.6 for details).

3.7 charges.bin / charges.dat

The file charges.bin contains the orbitally-resolved charges for each atom. In later versions of DFTB+ this format includes a check sum for the total charge and magnetisation. In the case of orbital potentials (p. 54) the file also contains extra population information for the occupation matrices.

This file is produced as part of the mechanism to restart ground state SCC calculations, see sections 2.5 and 2.3.8.

Equivalent data can also be present in the file charges.dat, but stored as plain text. The options WriteChargesAsText and ReadChargesAsText control which cases are generated and read respectively.

Appendix L contains details of the contents of the file.

3.8 md.out

This file is only produced for VelocityVerlet{} calculations (See p. 22). It contains a log of information generated during MD calculations, and appended every MDRestartFrequency steps. In the case of small numbers of atoms and long MD simulations it may be useful to set WriteDetailedOut to No and examine the information stored in this file instead.

3.9 Electrostatic potential data

The output from evaluating the electrostatic potential (see page 84). The first line consists of a comment mark followed by a logical variable as to whether there is an external electric field (or not), followed by 3 values for any regular grid pattern present in the system and the total number of points. If the data is gridded, the next four lines contain the origin and grid separation vectors in Ångstroms.

The next line is a comment, then the locations and the potential experience for a positive charge due to the internal field plus optionally the external field (from point charges or homogeneous electric fields). Values are given in Volts. In the case of gridded data, the location field is omitted.

For an example with a regular grid

```
# T    1    1    1 1
# 0.000000000000E+00 -0.200000000000E+01 -0.200000000000E+01
# 0.200000000000E+01  0.000000000000E+00  0.000000000000E+00
#  0.000000000000E+00  0.200000000000E+01  0.000000000000E+00
#  0.000000000000E+00  0.000000000000E+00  0.200000000000E+01
# Internal (V)        External (V)
  0.173386318927E-10  0.314737193575E+00
```

In the case where there is no regular grid:

```
#  T     0     0     0 1
#           Location (AA)             Internal (V)        External (V)
  0.0000E+00 -0.2000E+01 -0.2000E+01  0.173386318927E-10  0.314737193575E+00
```

In the case where data is generated for multiple geometry steps, this is also shown in the label:

```
#  F     1     5     5 25
#  0.000000000000E+00 -0.200000000000E+01 -0.200000000000E+01
#  0.100000000000E+01  0.000000000000E+00  0.000000000000E+00
#  0.000000000000E+00  0.100000000000E+01  0.000000000000E+00
#  0.000000000000E+00  0.000000000000E+00  0.100000000000E+01
# Internal (V) Geo 0
  0.215249473376E-01
   .
   .
# Internal (V) Geo 10
  0.215815549672E-01
   .
   .
```

3.10 Excited state results files

Several files are produced during excited state calculations depending on the particular settings from section 2.7.

Note: in the case of degeneracies, the oscillator strengths depend on arbitrary phase choices made by the ground state eigensolver. Only the sum over the degenerate contributions is well defined for most single particle transition properties, and label ordering of states may change if changing eigensolver or platform. For the excited state, properties like the intensities for individual excitations in degenerate manifolds again depend on phase choices made by both the ground and excited eigensolvers.

3.10.1 ARPACK.DAT

Internal details of the ARPACK solution vectors, see the ARPACK documentation [55] for details.

3.10.2 COEF.DAT

Data on the projection of this specific excited state onto the ground state orbitals. For the specific exited state, the (complex) decomposition of its single particle states onto the ground state single particle levels, together with its fractional contribution to the full excited state are given.

General format:

T F	Legacy flags
1 1.9999926523 2.0000000000	level 1, fraction of total WF, 2.0
-0.1944475716 0.0000000000 -0.1196876988 0.0000000000	real then imaginary projection of level 1 onto ground state 1, then ground state 2, etc.
-0.1196876988 0.0000000000 -0.1944475703 0.0000000000	
.	
.	
.	
2 1.9999866161 2.0000000000	level 2
-0.2400145188 0.0000000000 -0.1767827333 0.0000000000	real then imaginary projection of state 2
.	
.	
.	

3.10.3 EXC.DAT

Excitations data including the energies, oscillator strength, dominant Kohn-Sham transitions and the symmetry.

Example first few transitions for C_4H_4:

```
    w [eV]      Osc.Str.        Transition        Weight      KS [eV]    Sym.

 ================================================================

     5.551      0.5143882       11   ->    12        1.000       4.207      S
     5.592      0.0000000       10   ->    12        1.000       5.592      S
```

Two examples of singlet transitions with energies of 5.551 and 5.592 eV. The first is dipole allowed, the second not. In both cases they are transitions primarily (weight of 1.000) to single particle state 12, and are of singlet character (“S”).

In the case of spin-polarised calculations, an additional column of values are given instead of the symmetry, showing the level of spin contamination in the state (labelled as D<S*S>), with typically states where a magnitude of less than 0.5 is usually considered reliable [26].

3.10.4 SPX.DAT

Single particle excitations (SPX) for transitions between filled and empty single particle states of the ground state. These are given in increasing single particle energy and show the oscillator strength and index of the Kohn-Sham-like states that are involved.

```
      #      w [eV]      Osc.Str.        Transition

 ============================================
1     5.403     0.2337689     15  ->   16
2     5.403     0.2337689     14  ->   16
3     5.403     0.2337689     15  ->   17
4     5.403     0.2337689     14  ->   17
5     6.531     0.0000000     13  ->   16
6     6.531     0.0000000     12  ->   16
```

3.10.5 TDP.DAT

Detail of the magnitude and direction of the transition dipole from the ground to excited states.

3.10.6 TRA.DAT

Decomposition of the transition from the ground state to the excited states. The energy and spin symmetry are given together with the contributions from each of the single particle transitions.

3.10.7 NACV.DAT

Contains the TD-(LC)-DFTB non-adiabatic coupling vectors between many-body states,

$$d^{\xi}_{mn} = \langle \Psi_m | \frac{\mathrm{d}}{\mathrm{d}\xi} | \Psi_n \rangle \,.$$

The file is written as blocks, one for each requested electronic transition combination.

Each block starts with the indices of the two states being coupled, m and n (counting from 0 for the ground state), then the coupling vector in atomic units. Each line contains the coupling contribution associated with the three Cartesian displacements an atom in the geometry (combined index ξ), with one atom per line and ordered as in the geometry.

3.10.8 TEST_ARPACK.DAT

Tests on the quality of the eigenvalues and vectors returned by ARPACK. For the i^{th} eigen-pair, the eigenvalue deviation corresponds to the deviation from $(\langle \mathbf{x}_i | H | \mathbf{x}_i \rangle - \varepsilon_i)$, The eigen-vector deviation is a measure of rotation of the vector under the action of the matrix: $|(H|\mathbf{x}_i\rangle - \varepsilon_i|\mathbf{x}_i\rangle)|_2$, the normalisation deviation is $\langle \mathbf{x}_i | \mathbf{x}_i \rangle - 1$ and finally largest failure in orthogonality to other eigenvectors is given.

Example:

```
State  Ei deviation       Evec deviation    Norm deviation    Max non-orthog
1      -0.19428903E-15    0.80601119E-15    0.19984014E-14    0.95562226E-15
2      0.27755576E-16     0.85748374E-15    0.48849813E-14    0.36924443E-15
3      -0.12490009E-15    0.88607302E-15    0.88817842E-15    0.60384195E-15
```

3.10.9 XCH.DAT

Charges on atoms in the specified excited state. The top line contains the symmetry (Singlet or Triplet) and the number of the excited state. The next line is the number of atoms in the structure followed by some header text. Then on subsequent lines the number of each atom in the structure and its charge are printed.

3.10.10 XplusY.DAT

Expert file with the RPA $(X+Y)_{ia}^{I\Sigma}$ data for all the calculated excited states.

Line 1: number of single particle excitations and the number of calculated excited states
Line 2: Level number 1, nature of the state (S, T, U or D) then excitation energy (in Hartree)
Line 3: expansion in the KS single particle transitions
.
.
.
Line 2: Level number 2, nature of the state (S, T, U or D) then excitation energy (in Hartree)

3.10.11 XREST.DAT

Dipole moment of the specified excited state in units of Debye.

3.11 Electron dynamics results files

Real-time dynamics simulations produce the following output files:

3.11.1 energyvst.dat

Time-dependent energy components in Hartree. The column order is the following:

time (in fs), total energy, non-SCC energy, SCC energy, spin energy, external field energy, repulsive energy, nuclear kinetic energy, dispersion energy

3.11.2 qsvst.dat

Net atomic charges in electrons (negative atomic populations) as a function of time, written every `WriteFrequency` steps, with the following column order:

time (in fs), net total charge, charge (atom 1), charge (atom 2), ..., charge (atom N)

where N is the number of atoms, and the net total charge should always be zero (up to numerical presicion) for a neutral system.

3.11.3 atomenergies.dat

Atom-wise contribution to total energy (without nuclear kinetic energy) as a function of time, written every `WriteFrequency` steps, with the following column order:

time (in fs), total energy, energy (atom 1), energy (atom 2), ..., energy (atom N)

3.11.4 mu.dat/mux.dat/muy.dat/muz.dat

Dipole moment cartesian components μ_i as a function of time. The units of the quantities are given at the top of the file. The name depends on the type or perturbation: for kicks the polarisation direction of the Dirac-delta field is indicated in the name of the file; for lasers and pulses, the name is always mu.dat. The column order for a spin unpolarised calculation is the following:

time (in fs), μ_x, μ_y, μ_z

For (collinear) spin polarised calculations, the dipole components are also spin-dependent:

Time (in fs), μ_x (up), μ_y (up), μ_z (up), μ_x (down), μ_y (down), μ_z (down).

3.11.5 laser.dat

Created if Perturbation = Laser{} (see 2.8.1). Laser field components E_i (given in V/Å) as a function of time, with data in the following order:

time (in fs), E_x, E_y, E_z

3.11.6 molpopul1.dat/molpopul2.dat

Created if Populations = Yes. The number i in molpopul$i$.dat indicates the spin channel for spin polarised calculations (if spin unpolarised, only molpopul1.dat is written). It contains the populations projected onto the ground state molecular orbitals (GSMO), written every WriteFrequency steps. The projection is done by calculating first the change of basis matrix Λ from the coefficients matrix of the GSMO, $\Lambda = C^{-1}$, and then projecting the time-dependent density matrix of the system in the atomic orbital basis:

$$\rho^{GSMO}(t) = \Lambda \rho^{AO}(t) \Lambda^{\dagger}$$

The diagonal elements of ρ^{GSMO} are the sought populations. The first column is the time (in fs), and from the second one the populations starting from the lowest lying orbital.

3.11.7 forcesvst.dat

Time-dependent forces, calculated within the Ehrenfest approach and ignoring velocity-dependent terms, printed every WriteFrequency steps, if Forces = Yes. The file has $3N+1$ columns where N is the number of atoms, in the following order:

time (in fs), F_1^x, F_1^y, F_1^z, F_2^x, $\ldots$, F_N^x, F_N^y, F_N^z

3.11.8 tdcoords.xyz

Coordinates and velocities of the atoms saved every WriteFrequency steps in XYZ format, if Ion-Dynamics = Yes.

3.11.9 bondenergy.bin

Atom pair-wise non-self-consistent bond energy, written if WriteBondEnergy = Yes, every Write-Frequency steps. All numbers are stored in the following order (using the Fortran real precision that the code is compiled with):

time (in fs), $BE_{1,1}$, $BE_{2,1}$, $BE_{3,1}$, $\ldots$, $BE_{N,1}$, $BE_{1,2}$, $BE_{2,2}$, $\ldots$, $BE_{N,2}$, $\ldots$, $BE_{N,N}$

3.11.10 bondpop.bin

Atom pair-wise bond populations, written if WriteBondOrder = Yes, every WriteFrequency steps. (using the Fortran real precision that the code is compiled with):

time (in fs), $BP_{1,1}$, $BP_{2,1}$, $BP_{3,1}$, $\ldots$, $BP_{N,1}$, $BP_{1,2}$, $BP_{2,2}$, $\ldots$, $BP_{N,2}$, $\ldots$, $BP_{N,N}$

3.12 ppRPA_ener.DAT

Excitation energies obtained within the pp-RPA formalism (see section 2.7.2). This output file also includes the most dominant Kohn-Sham transition, its weight and energy difference as well as the spin multiplicity of the excited state.

Here are, for instance, the first three singlet-singlet transitions for furan:

```
   w [eV]       Transitions                    Weight        KS [eV]            Symm.

=================================================================

    6.411       HOMO -> LUMO + 0              0.998         5.162                 S
    6.904       HOMO -> LUMO + 1              0.973         6.755                 S
   11.339        HOMO -> LUMO + 0, 0          0.959         5.162,  5.162         S
```

The first two excitations are single, whereas the third one is a double transition with predominant HOMO-to-LUMO character.

3.13 REKS results files

Several files are produced during REKS calculations depending on the particular settings from section 2.9.

3.13.1 tdp.dat

Detail of the magnitude and direction of the transition dipole between all electronic states.

3.13.2 relaxed_charge.dat

Charges on atoms in the specified state. The top line contains the total charge of the system.

3.14 Halting DFTB+

In addition to stopping on successful completion, if there errors in the input, or if the limits for the iterative stages of a calculation are exceeded (for example exceeding MaxSccIterations or MaxSteps), DFTB+ can be externally halted manually.

During operation if files called stop_scc or stop_driver are present in the directory, DFTB+ will halt at the next SCC or geometry step respectively. If these files are present at the start of the calculation, DFTB+ will halt immediately (hence these files should be removed manually if used to halt the code).

Chapter 4

Transport calculations

Non-equilibrium Green's function (NEGF) calculations are now available with DFTB+. Within this formalism it is possible to treat quantum mechanical systems with open boundary conditions, i.e. systems connected to external reservoirs and therefore quantum transport. A new specific Transport{} block has been added to specify the geometry of such transport problems. Additional solvers have been added to the Solver section to either fully solve the open boundary problem (using the keyword GreensFunction{}) or the transmission through the system (TransportOnly). Finally a real-space Poisson solver is available for self consistent charge calculations and electrostatic gates (within a new section, Electrostatics, using the keyword Poisson{}).

4.1 Definition of the geometry

The input geometry for transport calculations can be a little tricky. In comparison to cluster or supercell boundary conditions, the geometry for transport calculation must also contain information about the contacts (external reservoirs). The contacting leads (or surfaces) are actually semi-infinite structures, supporting travelling waves. Unlike finite structures, where mo stationary current is possible, travelling waves can only exist in such open systems. The simulation is therefore partitioned into a device region and one or more contact regions.

note:
A single contact can be used to model semi-infinite surfaces or the ends of nanowires/tubes. A minimum of two contacts is required to simulate devices and to evaluate properties such as current flow. For SCC calculations, the potential shifts within contacts are stored in files. See appendix L.0.2 for the internal format of these files.

4.1.1 Rules to build a valid input geometry

1. All device atoms must come first in the structure.
2. Each contact must comprise of two subsequent unit cells, called principal layers (PLs). The two PLs together give all information about the contact structure and in the following are referred generally as a "contact".
3. A PL is a unit cell of the contacting lead that has interactions only with its nearest neighbour PLs in tight-binding terms (i.e. the Slater-Koster interactions only extend into immediately neighbouring PLs).

4. The ordering of the atoms within the two PLs of a contact must be consistent, in the sense that the two PLs must be exact periodic replicas of each other: If each PL comprise N atoms, the i^{th} atom in the first PL must have a corresponding identical atom, $i+N$, in the second PL which is related by translation to the position of atom i.

5. The first PL in a contact should be always the one which is closer to the device region.

6. All blocks should be contiguous in the structure and each atom must belong to one and only one region.

7. The geometry can be defined as a cluster or a supercell. In the first case is it understood that the contacts are one-dimensional wire leads.

8. If a structure is defined as being a *supercell*, only the lattice vectors that are transverse to the transport direction are meaningful. The periodicity specified along the transport direction is treated as a dummy vector (but must be present).

9. For each contact the periodicity along the transport direction is actually deduced from the separation between the two PLs (using the coordinate difference $\mathbf{r}(i+N)-\mathbf{r}(i)$). We refer to this vector as aligned along the *contact direction*.

10. All lattice vectors (including the contact direction vector) must be aligned parallel to one of the Cartesian axes x, y or z. In practice only rectangular cells are allowed in transport calculations at present.

An example of a non-periodic device with contacts attached is shown in Figure 4.1.

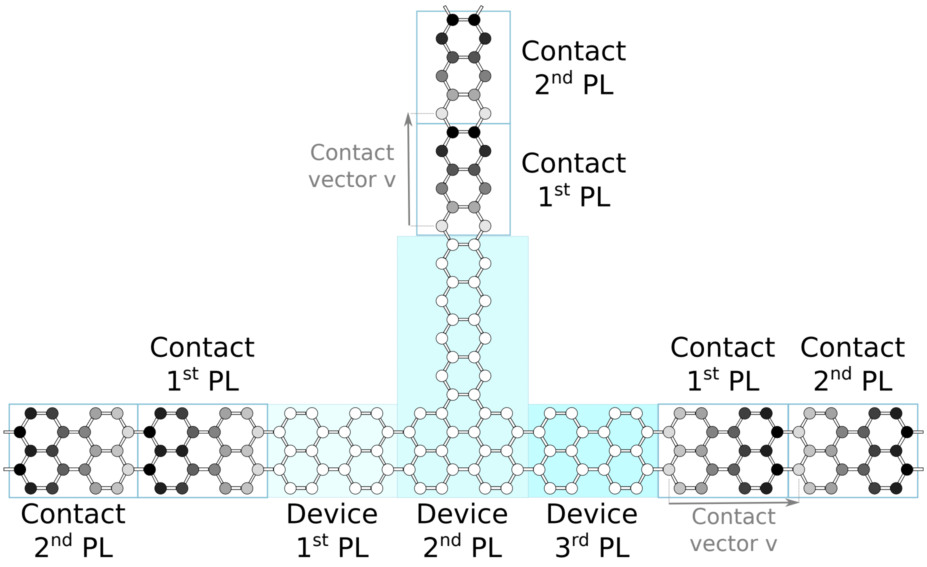

Figure 4.1: Example of a valid 3 contact device with principal layers marked.

Note:
The code *does not* currently check: if the device regions are consistently defined (rules 1 and 6); if the PL defined are really PLs (rule 3); or if the first PL defined is really the one closest to the device (rule 5).
The code *does* check rules 4, 8, 9 and 10. The check for rules 4 and 9 is performed on the atomic coordinates, such that

$$\mathbf{R}^2_{i+N} = \mathbf{R}^1_i + \mathbf{v} \qquad \forall i \in PL \tag{4.1}$$

where $\mathbf{R}_i^2$ are atomic coordinates of atoms in the second PL of the contact, $\mathbf{R}_i^1$ are atomic coordinates of atoms in the first PL and $\mathbf{v}$ is the contact lattice vector. The equality is verified within an accuracy that can be set by the user (see below for PLShiftTolerance).

Please take care when building structures and to cross-check them. Also consider looking at the examples distributed with the code. The input structure is often the first suspect when there are problems in transport calculations.

4.2 Transport

The Transport section collects together the information needed whenever open boundary conditions are used. It contains the description of the partitioning of the system into a *device* and the *contact* regions and additional information needed to calculate the required self-energies associated with the contacts. The transport block contains the following properties:

Device	p	-	117
Contact	p	-	118
Task	m	UploadContacts	119

An example transport geometry specification looks like:

```
Transport {
  Device {
    AtomRange = 1 8
  }
  Contact {
    Id = "source"
    AtomRange = 9 24
  }
  Contact {
    Id = "drain"
    AtomRange = 25 40
  }
}
```

Where the associated atomic geometries follow the rules of Section 4.1. In this specific example, there is only one principal layer in the device, but each contact contains two principle layers (atoms 9–16 and 17–24 in the “source” contact, atoms 25–32 and 33–40 in the “drain” contact).

4.2.1 Device{}

The Device blocks contains the following properties:

AtomRange	2i	-	117
FirstLayerAtoms	i+	1	

AtomRange defines the first and last atom of the device region.

FirstLayerAtoms defines the first atom of PLs in the device region. By default there is only one layer (the entire device region). Alternatively the user can manually reorder and group the atoms in the structure into distinct layers for more efficient Green's function calculations.

The device layers, unlike the contact PLs, do not need to represent unit cell repetitions. If the device geometry has specified principal layers, these must be ordered in such a way that all the atoms within each of the layer are contiguous in space and adjacent layers must be placed next to each other in the structure. This ensures that the constructed hamiltonian and overlap are block tri-diagonal. Refer to [74] for a description of the efficient iterative Green's function algorithm that can then be applied.

4.2.2 Contact{}

The contact block contains the following properties:

Id	s		
AtomRange	2i		
PLShiftTolerance	r		1E-5
Temperature	r		0.0
FermiLevel	r		
Potential	r		0.0
WideBand	l		No
LevelSpacing	r	WideBand = Yes	0.735

The sections Device and Contact are used to define the atomic range of each region. The user can also assign a label (Id) to each contact that can be used later for cross referencing. In the section Contact the user can add a keyword that specifies the accuracy for the internal check of the PLs (tolerance for rule 4 of structures, i.e. that accuracy to which (4.1) must be satisfied).

Id Assign a text label to the contact (must be 50 or fewer characters).

AtomRange Defines the first and last atom of the device region. **Note** the contacts should be defined such that the atoms included in the range are in continuous increasing order in the structure.

PLShiftTolerance [*length*] Used to set the absolute accuracy used to check principal layer (PL) consistency (see above). The default is 10^{-5} atomic units. Please be aware that using a large values may hide errors due to an inconsistent definition of the contacts, therefore it should not be modified.

Temperature [*energy*] Specifies the electronic temperature of the contact (see a more detailed discussion after the section Änalysis¨).

FermiLevel [*energy*] Optional overriding of the Fermi energy that is specified in the appropriate contact shift file.

Potential [*energy*] Specifies any additional electrostatic potential applied to the contact. The natural units of this quantity are a (potential) energy.

WideBand Use the wide band approximation for the contact. If set to Yes, the surface Green's function of the contact is not explicitly calculated but is instead assumed to be local and constant according to a specified density of states.

LevelSpacing [*energy*] Specifies the inverse of the density of states (DOS) per atom to be used for the Wide Band approximation. As an example, the DOS of gold at the Fermi level is 0.05 $\text{eV}^{-1}\text{atom}^{-1}$, which corresponds to an energy spacing of 20 eV ≈0.735 Hartree (the default value).

4.2.3 Task = ContactHamiltonian{}

The Task option is used to define which type of calculation should be performed. Before performing a transport calculation it is necessary to compute some equilibrium properties of the contacts by running a periodic boundary condition DFTB calculation. This necessary step must be carried out separately for each contact and can be done by specifying a Task=ContactHamiltonian block, as in the following example to calculate the source case.

```
Task = ContactHamiltonian {
  ContactId = source
  ContactSeparation [Angstrom] = 50.0
}
```

When Task=ContactHamiltonian the following options can be defined

ContactId	s	
ContactSeparation	r	1e3
WriteBinaryContact	l	Yes

ContactId Id label of the contact to be calculated.

ContactSeparation [*length*] Dummy separation in transverse direction (see the following explanation).

WriteBinaryContact Controls whether the contact shift file is written as a text file (file extension .dat) or a binary file (.bin).

The contact calculation computes the *bulk* Hamiltonian, self-consistent charges (if SCC) and Fermi level for each contact. This is a usual DFTB+calculation for which appropriate parameters must be included in the input file. For *supercell* structures the calculation of the contact is performed using corresponding supercells in which the transverse lattice vectors are those specified in the Geometry tag and the lattice vector along the *contact direction* is deduced from the PL separations (rule 9). If the structure is defined as a *cluster*, the contact calculation is performed for a *supercell* in which the contact is treated as one-dimensional periodic wire with a surrounding vacuum region. However, since DFTB+does not support pure one- and two-dimensional calculations, dummy lattice vectors are defined for the two remaining directions. The default value for these lattice vectors is 1000 a.u. (527 Å), which should guarantee sufficient wire to wire distances to avoid Coulomb interactions. The user can specify an alternative contact separation using the keyword ContactSeparation placed in the ContactHamiltonian block. Each contact computation produces one output file called shiftcont_ContactId.dat which storing energy shifts and Mulliken charges that must be present in the working folder in all subsequent transport calculations.

Note that during the contact calculation you will need to perform a k-point integration in the contact direction (as the contacts are semi-infinite). Whenever the system is defined as a cluster, DFTB+ will automatically extract the periodicity vectors from the geometry such that the first reciprocal vector will correspond to the contact direction. Therefore you must specify a k-point sampling for the periodic calculation by sampling along the first reciprocal lattice vector. As an example, if the structure is defined as a cluster (i.e., 1-dimensional wire leads), the source contact calculation will have an input file similar to:

```
...
Task = ContactHamiltonian {
  ContactId = source
}
...
Hamiltonian = DFTB {
...
KpointsAndWeights = SupercellFolding {
     8  0  0   # sampling points here regardless of the transport direction
     0  1  0
     0  0  1
     0.5 0.0 0.0
  }
}
```

On the other hand, if your structure is defined as a supercell (as an example, a molecule with bulk contacts) and the transport direction is along the *y* direction, your the source contact calculation will have an input file similar to:

```
...
Task = ContactHamiltonian {
  ContactId = source
}
...
Hamiltonian = DFTB {
...
KpointsAndWeights = SupercellFolding {
     4  0  0   # points in periodic direction
     0  8  0   # points in transport direction
     0  0  4   # points in periodic direction
     0.5 0.5 0.5
  }
}
```

This could seem confusing, but the underlining reasons is that in the cluster calculation the reciprocal lattice is set up by the code itself, while in the periodic calculation is set up by the user, who can chose any arbitrary direction. Refer to the transport cookbook and to the distributed examples for further clarification.

4.2.4 Task = UploadContacts{}

After the contact calculations are completed, it is now possible to perform actual transport calculations. This is activated simply specifying Task = UploadContacts, without additional options

(**Note** if no task is specified, DFTB+ assumes UploadContacts is the required task in the transport block). If you require potential shifts of the contacts, these should also be set for each contact.

```
Transport {
  Device {
    AtomRange = 1 8
  }
  Contact {
    Id = "source"
    AtomRange = 9 24
    # No specified shift in contact, effectively:
    # Potential = 0.0
  }
  Contact {
    Id = "drain"
    AtomRange = 25 40
    Potential = 1.0
  }
  Task = UploadContacts
}
```

Note: During the transport calculation you will not need to set up the k-point integration when the structure is defined as a cluster, just as in a regular DFTB+ calculation. For supercell calculations, integration perpendicular to the transport direction will need to be accurate, but the sampling grid can in the transport direction itself can have only a single value. In the special case where your device is a supercell but also wire-like, with a vacuum region lateral to its transport direction, the Gamma-point can be chosen:

```
  KPointsAndWeights = {
    0 0 0  1.0
  }
```

When Task=UploadHamiltonian the following options can be defined

ReadBinaryContact	l	Yes

ReadBinaryContact Controls whether the contact shift file should be read as a text file (file extension .dat) or a binary file (.bin).

4.3 GreensFunction

For calculations in open systems, instead of calculating the eigenstates of the system, a Green's function method is used to obtain the density matrix of the system. The Green's function (GF) solver can also be used for conventional supercell/cluster boundary conditions if required.

In order to activate Green's function calculations the user must define the keyword Solver = GreensFunction in the Hamiltonian section. The GF solver, either under equilibrium (no bias applied) or under non-equilibrium conditions, builds the density-matrix of the device region such that it is consistent with any contacts that are present. Strictly speaking the GF does not solve for the eigenstates of the system, however it logically substitutes the traditional construction of the density matrix from the eigenstates of the system, as would be obtained after the diagonalisation step. The usual DFTB+ self-consistent calculations can be driven using the GF solver.

The following table gives the important parameters of the solver:

Name	Type	Condition	Default	Page
Delta	r		1E-5	
ContourPoints	2i		20 20	
LowestEnergy	r		-2.0	
FermiCutoff	i		10	
EnclosedPoles	i		3	
RealAxisStep	r	RealAxisPoints=undefined	6.65E-4	
RealAxisPoints	r	RealAxisStep=undefined		
SaveSurfaceGFs	l		Yes	
ReadSurfaceGFs	l		No	
FirstLayerAtoms	i+	Transport = undefined	1	
FermiLevel	r	Transport = undefined		
LocalCurrents	l		No	

Note: For efficient GF calculation the device region must be partitioned into layers whose fundamental property is to interact with nearest-neighbour layers only (see section 4.1).

Delta [*energy*] A small positive imaginary delta used in the GF definition and required for the x contour integration.

ContourPoints The number of points along the complex contour integration of the GF along the segments $\mathscr{C}$ and $\mathscr{L}$ (see contour integration in section 4.5).

LowestEnergy [*energy*] The initial energy from which the integration starts. It should be low enough to ensure that all the electronic states are correctly included in the integration. The default is -2.0 Hartree (see contour integration).

FermiCutoff Integer number setting the Fermi distribution cutoff in units of kT. It is read only if the Fermi distribution temperature is greater than 0 (see contour integration).

EnclosedPoles The number of Poles enclosed in the contour. It is meaningful only in finite temperature calculations (see contour integration).

RealAxisStep [*energy*] The energy step along the real axis integration for non-equilibrium calculations. Note: RealAxisStep and RealAxisPoints cannot both be defined at the same time.

RealAxisPoints The number of points along the real axis integration needed in non-equilibrium calculations. The default depends on the electronic temperature and bias. Note: RealAxisStep and RealAxisPoints cannot both be defined at the same time.

SaveSurfaceGFs As the SCC cycle usually needs to repeat the calculation of the Green's function at given energy points and as the surface Green functions do not change during the SCC cycle, this flag allows for saving the surface Green functions to disk and so save computational time on every SCC cycle after the first.

ReadSurfaceGFs Loads the surface Green's function from a file at the the first SCC cycle. Note that this operation only makes sense if the energy integration points are identical to the calculation used to generate the surface Green's function files. The code does not verify whether this condition is fulfilled. In general there is no need to modify the defaults for ReadSurfaceGFs and SaveSurfaceGFs.

FirstLayerAtoms As described in Device block. Can be specified only if no Transport block exists. **Note:** the Green solver can be used also to calculate the density matrix when there are no open boundary conditions, for example to take advantage of the iterative scheme in quasi-1d systems. In this case, a Transport block is not defined and therefore FirstlayerAtoms{} should be given in the GreensFunction block. Also, the Fermi level of the system must be known and provided to fill up the electronic states.

FermiLevel [*energy*] Required to set the Fermi level that is used by the Green's solver to fill up the electronic states. This should be set unless the Fermi level is already specified by the presence of contacts to the (device region) system.

LocalCurrents if set to Yes, local bond-currents are computed using the non-equilibrium density matrix. This task is currently limited to **non-periodic** systems. The output is placed in a file lcurrent_u.dat (or lcurrent_d.dat depending on spin). The files are arranged in a table in order of increasing neighbour distance,

Atom(i)	x	y	z	nNeighbours	j1	$I_{i,j1}$	j2	$I_{i,j2}$	j3	$I_{i,j3}$	...

This file can be processed using the small code flux provided in tools/transport that helps in building plots for jmol.

GreensFunction section example:

```
Solver = GreensFunction {
  FirstLayerAtoms = 1 61 92 145
  Delta [eV] = 1E-4
  ContourPoints = 20 20
  RealAxisPoints = 55
  LowestEnergy [eV] = -60.0
  FermiCutoff = 10
  EnclosedPoles = 3
}
```

4.4 Solver = TransportOnly

The GreensFunction block is used to solve the full self-consistent NEGF transport problem. However, the block TunnelingAndDos{} within the Analysis block (see below), can be used to calculate the transmission function according to the Landauer formula, without solving for the full density matrix. This can be applied even for calculations where the density matrix and charge densities are not computed. Similarly, in these cases the electrostatics block should be omitted (i.e. the Electrostatics = Poisson). The keyword Solver = TransportOnly is used to jump straight to the post-SCC analysis. **Note:** This option cannot be used together with calculations which require forces, including geometry relaxations or md calculations.

4.5 Contour integration

Much of the computational work for transport is in the integration of the energy resolved density matrix, as represented via the NEGF matrix. The integration is efficiently performed with a complex contour integration and a real axis integration, as shown in Figure 4.2 and discussed in references [2, 73, 74]. All integrations are performed with Gaussian quadratures and the number

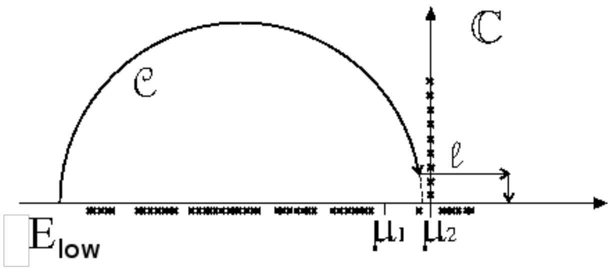

Figure 4.2: Contour integration in the complex plane for the Green's functions. The crosses represent poles of either G^r or the Fermi function.

of points must be specified manually. The complex contour integration is subdivided into two sections: the first section is the arc of a circle, $\mathscr{C}$, that can be computed with a few integration points (default 20); the second section is a line that intersects the contour and runs parallel to the real axis at a distance that depends on the number of poles of the Fermi function that are enclosed within the contour. Usually a good choice for the number of poles is between 3 and 5 (the default is 3). The poles are placed at the complex points $z_m = E_F + i(2m+1)\pi k_B T$ and therefore are separated from each other by $2\pi k_B T$, where k_B is Boltzmann's constant. At $T = 300$ K this corresponds to a separation of 156 meV. It should be noted that, as the temperature decreases, the separation between poles reduces. This makes the contour integration harder as it needs to walk across two singularities. At very low temperatures, $T = 10$ K, the separation is 5.2 meV. Below this temperature, the contour integration is treated as $T = 0$ in order to avoid numerical inaccuracies. The integration along the segment $\mathscr{L}$ extends up to $Re[z] = E_F + nk_B T$, where n is an integer number specified by the keyword FermiCutoff and has a default value of 10. In the limit $T = 0$ K the poles collapse into a non-analytic branch cut and the contour needs to be changed such that the second section of the complex contour becomes the arc of circle closing on the real axis. Finally, the real axis integration extends between the lowest and highest chemical potentials. The number of quadrature points should depend on the bias itself and can be set using RealAxisPoints or RealAxisStep. The default value is 1 pt/0.018 eV (actually 1500 pt/1 Hartree). In finite temperature calculations the segment is extended to include the Fermi cutoff by nk_BT on both sides $(\mu_1 - nk_BT, \mu_2 + nk_BT)$. In this case the number of quadrature points are increased by assuming the same point density defined in the range (μ_1, μ_2). Example: for a bias of 0.2 V, the default number of points is $0.2 \cdot 1500/27.21139 = 11$. At $T = 300$ K the interval is increased by 0.26 eV on both sides, therefore $0.26 \cdot 1500/27.21139 = 14.33$ which is truncated to 14 points, leading to a total of 38 points along the real axis. The use of the keyword RealAxisStep is usually more convenient because it ensures a consistent real axis integrations during, for example, a bias sweep.

Note: The GF solver can be used also for calculations other than the transport context. In cases where the position of the Fermi Energy is known with good accuracy, the density matrix solver based on the GF can be used to compute the electronic properties of clusters and supercells. The recursive algorithm may be an efficient solution to large problems having an elongated one dimensional shape.

4.6 Spin-polarised transport

Spin-polarised transport is available for collinear transport. Any contacts that are present should be calculated spin-polarized, even if their net spin is zero.

4.7 Poisson solver

The Poisson solver is a fundamental part of charge self-consistent non-equilibrium transport calculations and must be declared whenever an SCC NEGF calculation is performed using Electrostatics = Poisson. Under non-equilibrium conditions the self-consistent potential of the KS equations cannot be solved using the efficient γ-functional, but instead requires the definition of appropriate boundary conditions for the potentials imposed by the contacts. However, since the γ-functional is functionally related to a pure Hartree potential, it can be obtained in real space by solving a Poisson solver. The Poisson equation is solved in a *box* with hexahedral prism shape. This restriction is imposed by the Poisson solver being employed. This restricts calculations of supercell structures to orthorhombic super-lattices. An additional restriction is that the box sides must be aligned with the Cartesian axes, x, y, z.

Name	Type	Condition	Default	Page
Verbosity	i		51	
PoissonBox	3r			
BoxExtension	r		0.0	
MinimalGrid	3r		0.5 0.5 0.5	
PoissonAccuracy	r		1E-7	
AtomDensityTolerance	r		1E-6	
AtomDensityCutoff	r		14.0	
CutoffCheck	l		Yes	
NumericalNorm	l		No	
SavePotential	l		No	
PoissonAccuracy	r		1E-6	
MaxPoissonIterations	i		60	
BuildBulkPotential	l		Yes	127
ReadOldBulkPotential	l		Yes	127
OverrideDefaultBC	m		none{}	127
OverrideBulkBC	m		none{}	127
BoundaryRegion	m		global{}	127
Gate	m		none{}	129
MaxParallelNodes	m		none{}	130
RecomputeAfterDensity	l		No	
PoissonThickness	r	contacts = 1		

Verbosity This parameter controls the level and amount of output messages and takes values ranging from 1 to 100.

PoissonBox [*length*] Dimension of the Poisson box along the directions x, y and z.

BoxExtension [*length*] With this value it is possible to tune the position of the box interface between the device and contacts. By default (BoxExtension=0.0) the boundary is placed at the midpoint between the last device atom and the first contact atom.

MinimalGrid [*length*] The minimal requested grid spacing along x, y and z. The actual grid spacing chosen by the multigrid solver will be lower than this.

AtomDensityTolerance In order to calculate the potential, the Mulliken charges are projected on the real space grid. This parameter defines the cutoff after which the charge is considered to vanish (i.e., the spatial extension of the projected charge). The default is 1E-6 e. Note that the contacts must be at least twice the length over which a projected Mulliken charge extends. If this conditions is not fulfilled and CutoffCheck is set to Yes, the code will exit with an error message. Setting this parameter to a lower value could allow shorter contacts to be defined in some cases. However this could lead to error in the potential and hence to spurious reflections, therefore it should be left at its default value (or changed very carefully).

AtomDensityCutoff [*length*] Defines the atomic radius cutoff. This is an alternative to AtomDensityTolerance and directly specifies the distance over which charge density associated with an atom is considered to vanish.

CutoffCheck If set to No, consistency between contact length and charge extension is not verified (see AtomDensityTolerance and/or AtomDensityCutoff). The default is Yes. As with AtomDensityTolerance, this parameter should not be changed unless you know exactly what you're doing.

SavePotential Save the electrostatic potential to the file potential.dat and the charge density to charge_density.dat. Additional files Xvector.dat, Yvector.dat, Zvector.dat and box.dat are also created. These files can be converted to a cube file that can be visualised in jmol. See section 4.13 about transport tools.

PoissonAccuracy Defines the accuracy for the approximate solution of the Poisson equation (default value 10^{-6}).

MaxPoissonIterations Defines the maximum number of iterations allowed for the solver.

RecomputeAfterDensity When set to Yes, Poisson's equation is solved again after the density matrix is created in order to make the electrostatic energy consistent with the newly updated charges. In transport calculations it is set to No by default in order to avoid the extra time spent on the Poisson step. This does not affect the SCC loop or other calculations apart from the electronic energy and forces.

PoissonThickness In the special case of a single contact (cases like the end of semi-infinite wires or surfaces of crystals), the thickness of the Poisson box normal to the surface of the contact can be set with this command.

Note: The Poisson box can be specified using the keyword PoissonBox. In calculations where two contacts face each other along the same axis, setting the box-size along this axis will has no effect (the code adjusts to the correct size internally). The PoissonBox keyword is redundant (and should not be specified) when the system is periodic, since in this case the box geometry is taken from the supercell lattice vectors.

Numerical error in the potential will results in spurious discontinuities at the contact-device interfaces. The default tolerances should be sufficient to avoid this in most cases.

Below is a a typical example of the whole Poisson block specification. Some of the keywords are described in the next subsections.

```
Electrostatics = Poisson {
 PoissonBox [Angstrom] = 20.0 20.0 20.0
 MinimalGrid [Angstrom] = 0.3 0.3 0.3
 SavePotential = No
 BuildBulkPotential = Yes
 ReadOldBulkPotential = No
 BoundaryRegion = Global {}
 PoissonAccuracy = 1E-7
 Gate = Planar{
    GateLength_l [Angstrom] = 10.0
    GateLength_t [Angstrom] = 20.0
    GateDistance [Angstrom] = 7.0
    GatePotential [eV] = 1.0
 }
}
```

4.7.1 Boundary Conditions

The Poisson equation is solved imposing boundary conditions (BC) on the potential at the six faces of the Poisson Box. In transport calculations for non-supercell geometries comprising two contacts placed along the same axis, the BCs are chosen as follows:

Dirichlet	fixed potentials on the two contact faces with values defined by the applied potentials (see UploadContacts).
Neumann	zero normal field on the remaining 4 lateral box faces

In periodic supercells the BCs are: **Dirichlet** (fixed potentials) on the two contact faces with values defined by the applied potentials (see UploadContacts) and **Periodic** on the remaining 4 lateral box faces.

In some specific cases Neumann BCs can be set on one contact. In order to do so it is necessary to use OverrideDefaultBC (see below).

The device and contact potentials should smoothly join at the interface. In order to achieve this the code computes the bulk potential of each contact and uses the result as a BC on the contact face of the Poisson box. This is useful when the contact potential is not uniform due to charge rearrangements. Any externally applied contact potential (Potential) is added to the bulk potential. The user can deactivate this calculation by setting the keyword BuildBulkPotential to No.

Note: The bulk potential is computed on a special box that has “lateral” sizes copied from the device box, and has the size of one PL along the contact direction. The BCs are—so to speak—inherited from the device region. In particular:

1. Along the contact direction periodic BCs are imposed on both faces.
2. On the other four faces the BCs are copied from the device region (supercell or cluster).
3. The user can override this setting using OverrideBulkBC (see below).

4. When all four faces inherit Neumann BC (default for the device region), these are *ALL* internally changed to Dirichlet, because the solver cannot handle this situation as it gives rise to a singular matrix.

BuildBulkPotential (default: Yes) is used to calculate the electrostatic potential of the contacts and the result is used as a Dirichlet boundary condition on the contact face (superimposed to the contact potential).

ReadOldBulkPotential Read a previously computed bulk potential from hard-disk.

BoundaryRegion Specifies how the Dirichlet boundary conditions are treated on each contact face of the Poisson box. It can be Global, Square or Circle. Global means that the BC is applied to the entire face of the box, whereas the other keywords imply that the Dirichlet BC are applied on a cross-section projected on the contact face. This is useful for instance when handling nanowire contacts, for which it is not really correct to impose a constant potential on the whole face of the Poisson box.

BufferLength [*length*] can be used to set the size of the boundary region beyond the atomistic size which is determined as the minimal circle or square containing all atoms of the contact cross-section.

Name	Type	Condition	Default	Page
BufferLength	r		9.0	

Example:

```
BoundaryRegion = Circle {
  BufferLength [Angstrom] = 3.0
}
```

In some special cases it might be necessary to override the default BCs applied by the code to the Poisson equation. Currently this can be done using the keywords: OverrideDefaultBC and OverrideBulkBC.

In the special case of a single contact, the boundary condition on the other side of the box to that contact is automatically over-ridden to be of Neumann type (but can still then be over-ridden with OverrideDefaultBC).

OverrideDefaultBC block is used to override the BCs described above. It can be used to force Dirichlet or Neumann BCs along some specified directions or on one of the four lateral faces of the Poisson box.

Boundaries is used to specify on which face different BCs must be imposed. Assuming contacts are aligned along z, the keyword can be set to be any of xmin, xmax, x, ymin, ymax or y.

```
OverrideDefaultBC = Dirichlet {
   Boundaries = xmin
}
```

For instance, setting a Dirichlet BC on Boundaries = xmin imposes $\phi(x,y,z) = 0$ on the face placed at $x = x_{\min}$, while boundaries = xmax would impose $\phi(x,y,z) = 0$ on the face placed at $x = x_{\max}$.

When Dirichlet needs to be forced on both faces it is possible to use either boundaries = xmin,xmax or simply boundaries = x. The same syntax can be used to impose conditions on more faces, using boundaries = x,y or boundaries = x,ymin.

A similar strategy can be used to impose different boundary conditions on the contacts. For instance, a Neumann BC can be set on one contact face by using

```
OverrideDefaultBC = Neumann {
    Boundaries = zmin
}
```

Note that the user should know which face of the Poisson Box corresponds to the desired contact. Furthermore, if the user sets Neumann at all contacts the Poisson solver will not converge (singular matrix) unless the Dirichlet condition is imposed somewhere else (e.g., a gate potential is present).

It is also possible to override the default BCs when computing the bulk potential.

OverrideBulkBC block is used to override bulk BC usually copied from the device region.

Boundaries has the same meaning and syntax as in OverrideDefaultBC.

For example by choosing

```
OverrideBulkBC = Neumann {
    Boundaries = x, y
}
```

4.7.2 Electrostatic Gates

The option Gate can be used to specify an electrostatic gate. Currently the available gate types are Planar and Cylindrical. There are some restrictions as the planar gate must be placed with its face parallel to the x-z plane, i.e., the gate direction must be along y. At the same time the transport direction should be along the z-axis (i.e. perpendicular to the gate). The latter is not really a restriction but it gives meaning to "longitudinal" (l) and "transverse" (t) in the geometrical definitions of the gate lengths. Example:

```
Gate = Planar {
   GateLength_l [Angstrom] = 20.0
   GateLength_t [Angstrom] = 20.0
   GateDistance [Angstrom] = 7.0
   GatePotential [eV] = 1.0
}

Gate = Cylindrical {
   GateLength [Angstrom] = 10.0
   GateRadius [Angstrom] = 7.0
   GatePotential [eV] = 1.0
}
```

The various options for the gates have the following meanings:

GateLength_l [*length*] Sets the gate length along the transport direction (always assumed to be *z*). The gate is centred in the middle of the device region.

GateLength_t [*length*] Sets the gate extent transverse to the transport direction (assumed to be *x*). The gate is centred in the middle of the device region.

GateDistance [*length*] Sets the distance of the gate from the centre axis of the device region.

GatePotential [*energy*] Sets the potential applied to the gate.

GateRadius [*length*] For a cylindrical gate, sets the distance of the gate from the centre axis or gate radius.

Name	Type	Condition	Default	Page
GateLenth_l	r		0.0	
GateLenth_t	r		0.0	
GateDistance	r		0.0	
GatePotential	r		0.0	
GateRadius	r		0.0	

Note that the gate option has not be tested thoroughly and may still contain bugs. Please report any problems you encounter to the developers.

4.8 Parallelisations

The code has been parallelised in two main parts. The Non-equilibrium Green's functions are computed by distributing the energy points along the contour and real axis calculations. Contour and real axis integrations are independent and separately distributed. Load balancing has to be taken care of by the user. For instance if ContourPoints = {20 20} (i.e. 40 in total) and RealAxisPoints = 60, by setting 10 MPI nodes, each node will handle 4 points along the contour and 6 points along the real axis.

Mixed OpenMP/MPI calculations are possible. When compiling DFTB+ the user should link against threaded BLAS/MKL, rather than sequential. Numerical experiments show that best performance on multicore CPUs is generally obtained by running independent MPI processes on physical sockets and exploiting OpenMP multithreading within each socket. For instance NEGF can exploit threaded matrix-matrix products. The user can experiment by setting the environment variable OMP_NUM_THREADS.

The Poisson solver itself has not been parallelised yet. Currently the assembly of the charge density on the real-space grid and the projection of the potential onto the atoms has been parallelised. Since the gathering of the charge density on each node can easily hit communication bottlenecks, the user can use the parameter MaxParallelNodes to control distributions of these calculations. The default is MaxParallelNodes=1, this can be increased until speedups are observed.

MaxParallelNodes	i		1	

4.9 Analysis

The Analysis block is used to specify post-scf calculations such as tunnelling or projected DOS.

```
Analysis{
  TunnelingAndDOS{
    EnergyRange [eV] = {-5.0 -3.0}
    EnergyStep [eV] = 0.02
  }
}
```

4.10 TunnelingAndDos

This method block can be specified in Analysis 82 and it is used to calculate the transmission by means of the Caroli formula, the current by means of the Landauer formula and the density of states from the spectral function. This block can only be specified if an open boundary conditions system has been defined in Transport (see p.117).

EnergyRange	2r		
EnergyStep	r		
TerminalCurrents	p		
ContactTemperature	Nr	T_{elec}	
Region	p		131
computeLDOS	l	Yes	
WriteTunn	l	Yes	
WriteLDOS	l	Yes	

EnergyRange [*energy*] Contains the energy range over which the transmission function and local density of states are computed.

EnergyStep [*energy*] Is the energy sampling step for evaluating properties.

TerminalCurrents{} in multi-terminal configurations is used to define the terminal across which current must be computed. The terminal pairs are defined by using the keyword EmitterCollector, for example:

```
TerminalCurrents{
  EmitterCollector = {"source" "drain"}
  EmitterCollector = {"source" "gate"}
}
```

The block TerminalCurrents may be omitted since the code automatically sets all possible independent combinations for the terminal currents. For example in a 4-contact calculations the currents are 1–2, 1–3, 1–4, 2–3, 2–4 and 3–4.

ContactTemperature [*energy*] Specifies the electronic temperature for the contacts used in the calculation of currents. It expects an array of real values, one per contact, which following the order the contacts are listed in Transport.

Region{} This block defines atomic ranges or orbitals on to which the local density of states is projected. The definition in the block follow the same syntax as a DFTB+ calculation without transport (see section 2.6.1). If Region is absent the default behaviour is to compute the projected DOS on the whole central region.

ComputeLDOS Change the default bahaviour of projected DOS calculation. When set to No, PDOS is not computed. In the case of two contacts the code can exploit a fast and less memory consuming algorithm for transmission.

WriteTunn The transmission coefficients are written also to a separate file for quick reference. If set to No, the transmission coefficient are only written to DFTB+ output files (detailed.out and detailed.xml, autotest.tag).

WriteLDOS same as above, but for the density of states.

4.11 Setting electronic temperature

In the current state of the code the electronic temperature of the system can be set in different places. One place is within the Hamiltonian block in the Filling section. The Temperature specified here is effective for the whole device and applies to all contact calculations as well. Note that during contact calculations the temperature is read from the Filling section and *not* from the contact section. During contact calculations only Fermi filling can be used.

When a temperature is specified in the Contact section of the Transport block, it overrides the system temperature specified in filling.

Note that the present electronic behaviour for transport is going to be changed soon.

It is also possible to specify a (different) temperature in the section TunnelingAndDOS, within the block Analysis. The latter applies only in the calculation of currents (integration of the transmission function).

Although slightly inconsistent, in some cases it is useful to be able to set a somewhat larger temperature in the calculation of the density than used for property calculations (e.g. currents), as this helps the convergence of the self-consistent loop.

4.12 Troubleshooting transport

The DFTB+ transport machinery is designed to calculate transport in structures with a large number of atoms. To take full advantage of the iterative algorithm, be sure that the system is correctly partitioned into Principal Layers, as described in section 2.6.1. Be aware that an incorrect partitioning will lead to wrong results. If you are not completely confident, you can run a calculation on a test system with and without principal layer partitioning in the device region and the results *should* be the same.

On some systems, a Segmentation Fault error could occur while running relatively large structures. This can happen because the stack memory limit on your system has been exceeded (the Intel compiler for example can show this behaviour). You can troubleshoot this by setting a higher limit for the stack memory. In bash you can remove the stack memory limitation with the command line `ulimit -s unlimited`.

4.13 Transport Tools

Some tools useful for transport calculations can be found in tools/misc/transport.

buildwire

This tool can be used to create a one-dimensional nanowire, ready for transport calculations. A Principal Layer must be defined as a gen file, complete with supercell information. The code needs as input the number of PLs in the device region and the direction of the device. The resulting geometry will include 2 PLs for each contact and the specified number of PL repeats in the device region.

flux

This can be used to visualise the local bond currents in a junctions. The code reads the output files lcurrents.dat and writes out a script for jmol with arrows of different length/thickness for the currents.

makecube

This program can be used to convert the real-space potential.dat or charge_density.dat files computed on the Poisson box to a cube file that can be plotted using jmol.

```
 makecube potential.dat [-r refpot] [-b boxfile xfile yfile zfile]
```

Options:

-r refpot provides a reference potential that is subtracted from potential.dat For instance it is possible to subtract the equilibrium potential from the bias cases.

-b The code reads by default the files box.dat, X,Y,Zvector.dat, but different filenames can be given with this flag

Once the cube file has been created it can be read into jmol and visualised using the following script,

```
script colormap128.jmol
load "structure.xyz"
isosurface pl1 fullplane plane {-1.2 0.8 0 0} color range all colorscheme 'user' 'potential.cube'
```

Notice that a 128 palette colour map is provided in the tool folder. Also note that the structure should be converted to xyz to be read into jmol.

Chapter 5

MODES program

The MODES program calculates vibrational modes using data created by DFTB+.

5.1 Input for MODES

The input file for MODES must be named modes_in.hsd and should be a Human-friendly Structured Data (HSD) formatted file (see Appendix B) and must be present in the working directory.

The table below contains the list of the properties, which must occur in the input file modes_in.hsd:

Name	Type	Condition	Default	Page
Geometry	plm		-	10
Hessian	p		{}	136

Additionally optional definitions may be present:

Name	Type	Condition	Default	Page
DisplayModes	p		-	137
Atoms	i+lm		1:-1	
WriteHSDInput	l		No	
RemoveTranslation	l		No	
RemoveRotation	l		No	
SlaterKosterFiles	plm		-	
Masses	p			138
BornCharges	p		{}	136
BornDerivs	p		{}	137

Geometry Specifies the geometry for the system to be calculated. See p. 10.

Hessian Contains the second derivatives matrix of the system energy with respect to atomic positions. See p. 136.

SlaterKosterFiles Name of the Slater-Koster files for every atom type pair combination. See p. 50. This is used to obtain the masses, so if these are explicitly set using Masses{}, it is not required.

DisplayModes Optional settings to plot the eigenmodes of the vibrations. See p. 137.

Atoms Optional list of atoms, ranges of atoms and/or the species of atoms for which the Hessian has been supplied. *This must be equivalent to the setting you used for MovedAtoms in your* DFTB+ *input when generating the Hessian.*

WriteHSDInput Specifies, if the processed input should be written out in HSD format. (You shouldn't turn it off without good reason.)

RemoveTranslation Explicitly set the 3 translational modes of the system to be at 0 frequency.

RemoveRotation Explicitly set the rotation modes of the system to be at 0 frequency. Note, for periodic systems, this is usually incorrect (if used for a molecule full inside the central cell, it may be harmless).

Masses If present, replace the atomic masses from the Slater-Koster files. See p. 138

5.1.1 Hessian{}

Contains the second derivatives of the total energy, see p. 21 for details of the DFTB+ options to generate this data. The derivatives matrix must be stored as the following order: For the i, j and k directions of atoms $1 \ldots n$ as

$$\frac{\partial^2 E}{\partial x_{i1} \partial x_{i1}} \frac{\partial^2 E}{\partial x_{j1} \partial x_{i1}} \frac{\partial^2 E}{\partial x_{k1} \partial x_{i1}} \frac{\partial^2 E}{\partial x_{i2} \partial x_{i1}} \frac{\partial^2 E}{\partial x_{j2} \partial x_{i1}} \frac{\partial^2 E}{\partial x_{k2} \partial x_{i1}} \cdots \frac{\partial^2 E}{\partial x_{kn} \partial x_{kn}}$$

Note: for supercell calculations, the modes are currently obtained at the $\mathbf{q} = 0$ point, irrespective of the k-point sampling used.

5.1.2 BornCharges{}

If the mixed second derivatives of the energy with respect to electric field and position are available ($Z^\star$ values), these can be used to gain an estimate of the infrared activitiy of the vibrational modes. The resulting transition strengths (in arbitrary units) are then printed. The derivatives are also equivalent to the first derivatives of the dipole moment with respect to atomic positions, or the derivatives of forces with respect to external electric field.

The Born charges can be generated by DFTB+ by evaluating finite difference derivatives (see sec. 2.3.7), at the same time that the hessian matrix is calculated. The resulting Born charges are stored in the file *born.out*.

The Born charge data is ordered in the same way as the hessian information:

$$\begin{array}{r}
\frac{\partial \mu_x}{\partial x_{i1}} \frac{\partial \mu_y}{\partial x_{i1}} \frac{\partial \mu_z}{\partial x_{i1}} \\
\frac{\partial \mu_x}{\partial x_{j1}} \frac{\partial \mu_y}{\partial x_{j1}} \frac{\partial \mu_z}{\partial x_{j1}} \\
\frac{\partial \mu_x}{\partial x_{k1}} \frac{\partial \mu_y}{\partial x_{k1}} \frac{\partial \mu_z}{\partial x_{k1}} \\
. \\
. \\
. \\
\frac{\partial \mu_x}{\partial x_{kn}} \frac{\partial \mu_y}{\partial x_{kn}} \frac{\partial \mu_z}{\partial x_{kn}}
\end{array}$$

The input to read the Born charges is:

```
BornCharges = {
  <<< born.out
}
```

5.1.3 BornDerivs{}

If the mixed third derivatives of the energy with respect to electric field and position are available ($Z^\star$ values differentiated with respect to applied field), these can be used to gain an estimate of the Raman activity of the vibrational modes. As implemented, this is the change in the polarisability magnitude along the vibrational modes' directions, not the Raman intensity (see for example [82] for a full discussion of calculating Raman intensities).

The resulting transition strengths (in atomic units) are then printed.

The input to read the Born charge derivatives is:

```
BornDerivs = {
  <<< bornderiv.out
}
```

These derivatives can be evaluated with DFTB+ by calculating the second derivatives of the energy (see 2.3.7) while also requesting either the static or dynamic electric polarisability (see section 2.6.4). The resulting Born charges derivatives are stored in the file *bornderiv.out*.

5.1.4 DisplayModes{}

Allows the eigenvectors of the system to be plotted out if present

PlotModes	i+lm	1:-1
Animate	l	Yes

PlotModes Specifies list of which eigenmodes should be plotted as xyz files. Remember that there are $3N$ modes for the system (including translation and rotation).

Animate Produce separate animation files for each mode or a single file multiple modes where the mode vectors are marked for each atom.

Masses

Provides values of atomic masses for specified atoms, ranges of atoms or chemical species. This is useful for example to set isotopes for specific atoms in the system.

Mass	p

Any atoms not given specified masses will use the default values from the appropriate homonuclear Slater-Koster file. An example is given below:

```
Masses {
  Mass {
    Atoms = H
    MassPerAtom [amu] = 1.007825
  }
  Mass {
    Atoms = C
    MassPerAtom [amu] = 13.003355
  }
  Mass {
    Atoms = 1:2
    MassPerAtom [amu] = 2.014102
  }

}
```

where Atoms specifies the atom or atoms which each have a mass of MassPerAtom assigned.

Chapter 6

PHONONS

The tool PHONONS allows to perform quantum thermal transport calculations [83]. The quantities that can be computed are phonon transmission function and, hence, the thermal conductance. The non-equilibrium Green's function formalism for phonon transport is used. Currently only coherent transport is possible based on a dynamical matrix. Support for Dynamical Matrices other than DFTB+are possible. Phonon dispersion of materials can also be computed using this tool, based on a supercell geometry.

The input file must be named `phonons_in.hsd` and should be a Human-friendly Structured Data (HSD) formatted file (see Appendix B).

6.1 Phonon transport calculation

The input file has a similar structure to that defined for electron transport calculations and for the calculation of phonon modes (see Chapters 4 and 5).

The table below contains the relevant blocks in the input file,

Name	Type	Condition	Default	Page
Geometry	plm		-	10
Transport	plm		-	140
Masses	plm		-	
Hessian	p		{}	141
Analysis	plm		-	141

Geometry Specifies the geometry for the system to be calculated. Follow the steps described in Section 4.1 for a proper generation of the geometry file to compute transport properties. Also use the tool SETUPGEOM to help building a valid structure for transport (See Chapter 8).

Transport Contains the description of the partitioning of the system into a *device* and *contact* regions and additional information needed to calculate the required self-energies associated with the contacts. See Section 4.2 and p. 140.

Masses Specifies the mass of atoms in the geometry file. Default values are the standard atomic weights of the chemical elements. Alternatively it is possible to insert them manually or read thenm from the Slater-Koster files.

Hessian Contains information about the second derivatives matrix of the system energy with respect to atomic positions. See p. 141.

Analysis This block is used to introduce the parameters for the calculation of phonon transport properties: phonon transmission, (total and projected) density of states, and thermal conductance. See p. 141.

6.1.1 Transport{}

The block Transport is used to specify the partitioning of the structure into contacts and extended central region. The temperature of the contacts is added in the contact sections. An example of the transport geometry specification looks like:

```
transport {
  device {
    AtomRange = 1 50
  }
  contact {
    Id = "Drain"
    atomrange = 51 100
    temperature [K] = 300.0
  }
  contact {
    Id = "Source"
    atomrange = 101 150
    temperature [K] = 300.0
  }
}
```

Device{}

The device block contains the following properties

AtomRange	2i	-	140

AtomRange defines the first and last atom of the device region.

Contact{}

The contact block contains the following properties:

Id	s	
AtomRange	2i	
PLShiftTolerance	r	1E-5
Temperature	r	0.0

Id Assign a text label to the contact (must be 50 or fewer characters).

AtomRange Defines the first and last atom of the device region. **Note** the contacts should be defined such that the atoms included in the range are in continuous increasing order in the structure.

PLShiftTolerance [*length*] Used to set the absolute accuracy used to check principal layer (PL) consistency (see above). The default is 10^{-5} atomic units. Please be aware that using a large values may hide errors due to an inconsistent definition of the contacts, therefore it should not be modified.

Temperature [*temperature*] Specifies the temperature of the contact. For two contact calculations, if temperature of the contacts are different then a heat flux will be computed.

6.1.2 Hessian{}

The Hessian matrix is at the core of the calculation. It contains the second derivatives of the energy ,

$$\frac{\partial^2 E}{\partial x_{i1}\partial x_{i1}} \frac{\partial^2 E}{\partial x_{j1}\partial x_{i1}} \frac{\partial^2 E}{\partial x_{k1}\partial x_{i1}} \frac{\partial^2 E}{\partial x_{i2}\partial x_{i1}} \frac{\partial^2 E}{\partial x_{j2}\partial x_{i1}} \frac{\partial^2 E}{\partial x_{k2}\partial x_{i1}} \cdots \frac{\partial^2 E}{\partial x_{kn}\partial x_{kn}}$$

The Hessian matrix can be supplied by DFTB+, see p. 21 The derivatives matrix must be stored as the following order: For the i, j and k directions of atoms $1 \ldots n$. The tool PHONONS can also read other formats of Hessians computed with other codes. Currently Gaussian and CP2K are implemented.

The table below contains the list of properties which may be present in this block:

Cutoff	r	0.0
Matrix	p	{}

An example of the Hessian specification looks like:

```
Hessian {
  Cutoff = 50.0
  Matrix = dftb{}
}
```

Cutoff Specifies the range of interacting atoms in atomic units.

Matrix{} Contains the second derivatives of the energy. Setting `dftb` Matrix type reads the file 'hessian.out'. Currently the code can also support CP2K file format.

Note: for supercell calculations, the phonon transmission is obtained at the $\mathbf{q} = 0$ point, irrespective of the k-point sampling used to compute the Hessian.

6.1.3 Analysis{}

The analysis block looks like:

```
Analysis {
 TunnelingAndDOS{
    FreqRange = 1e-6 1e-2
    FreqStep = 4e-5
    DeltaModel = deltaOmega{
      Delta = 1e-4
    }
    Region = {
      Atoms = 1:50
    }
  }
  ModeType = longitudinal
  Conductance{
    TempRange [K] = 1.0 802.0
    TempStep [K] = 1.0
  }
}
```

TunnelingAndDOS{}

This method block is used to calculate the transmission by means of the Caroli formula in terms of the phonon Green's functions of the device and the self-energies of the contacts. The density of states is also computed from the spectral function. This block can only be specified if an open boundary conditions system has been defined in Transport (see p.117).

FreqRange	2r		
FreqStep	r		
DeltaModel	r		
Region	p	{}	142

FreqRange [*energy*] Contains the frequency range over which the transmission function and local density of states are computed.

FreqStep [*energy*] Is the frequency sampling step for evaluating properties.

DeltaModel Defines the frequency dependent model for δ in the Green's function. Possible values are 'deltaSquared', 'deltaOmega' and 'Mingo' model. The case 'deltaSquared' sets a constant value equal to Delta*Delta for dimensional reasons. The case 'deltaOmega' sets $\delta = \text{Delta} * \omega$. The Mingo model sets $\delta = f(\omega) * \omega^2$, where $f(\omega) = \text{Delta}(1 - \omega/\omega_{max})$. In principles this model, as suggested in [95], should be good in the limit $\omega \to 0$, however in some case it produces an extremely low value of δ that result in numerical problems. The 'deltaOmega' model is recommended.

Delta [*energy*] Defines the imaginary frequency value for the phonon Green's function. Note that in the Mingo model Delta is just a dimensionless scaling factor.

Wmax [*energy*] Defines the cutoff frequency typically set as the maximal phonon frequency of the problem.

Region{} This block defines atomic ranges on to which the density of states is projected. The definition in the block follow the same syntax as a DFTB+ calculation without transport (see section 2.6.1).

ModeType{} Defines the type of modes for which the transmission is restricted to. It is basic implementation that allows to project on mode components along cartesian axes. Currently available options are

- x-axis Projects on x-axis
- y-axis Projects on y-axis
- z-axis Projects on z-axis
- longitudinal Projects on z, assuming transport direction is along z.
- transverse Projects on x-y.
- inplane Projects on x-z, assuming a 2D planar structure on this plane.
- outofplane Projects on y, assuming a 2D planar structure on the xz plane.
- all Computes transmission for all modes.

Conductance{}

This method block is used to calculate the thermal conductance by means of the Landauer formula.

TempRange	2r
TempStep	r

TempRange [*temperature*] Contains the temperature range over which the thermal conductance is computed.

TempStep [*temperature*] Is the temperature sampling step for evaluating the thermal conductance.

6.2 Phonon dispersion

In order to compute phonon dispersions the following steps must be followed. 1. Start from a bulk cell and accurately relax setting a large number of k-points and tight MaximalForceComponent=1$e-5$. Allow LatticeOpt=Yes. 2. Generate a suitable supercell using repeatgen in dptools. This supercell must have the central cell at the beginning and then the repeated cells must follow. Only odd repetitions are possible such as 3x3x3 or 5x5x5. 4. Check the forces of this supercell are still very small. 5. Compute the Hessian for all atoms of this supercell. 6. Run PHONONS for bandstructure calculations using the settings below.

The relevant sections in the input file are:

Name	Type	Condition	Default	Page
Geometry	plm		-	10
PhononDispersion	plm		-	140
Masses	plm		-	
Hessian	p		{}	141

6.2.1 PhononDispersion{}

The PhononDispersion block looks like:

```
PhononDispersion{
  SuperCell = 3 3 3
  OutputUnits = cm
  KPointsAndWeights= Klines{
    1 0.5 0.5 0.5   # L
   10 0.0 0.0 0.0  # Gamma
   10 0.5 0.0 0.5  # X
  }
 }
```

Supercell Specifies the number of repetitions of the unit cell along the lattice vectors. This must be the same as those set in repeatgen to generate the supercell used for phonon calculations. Usually 3x3x3 supercells or in case of low-dimensional structures can be for instance 3x3x1 (2-dimension).

KPointsAndWeights{} Defines the k-point path for the phonon dispersion. The syntax is the same as that defined in the DFTB+ input.

OutputUnits Specifies the output units of the phonon bandstructure. These can be one of 'H', 'eV', 'meV', 'cm' or 'THz'.

Chapter 7

WAVEPLOT

The WAVEPLOT program is a tool for the visualisation of molecular orbitals. Based on the files created by a calculation performed by DFTB+ it is capable of producing three dimensional information about the charge distribution. The information is stored as cube files, which can be visualised with many common graphical tools (e.g. VMD or JMOL).

The user controls WAVEPLOT through an input file, choosing which orbitals and charge distributions should be plotted for which spatial region. Since the information about the shape of the basis functions is usually not contained in the Slater-Koster files, the coefficients and exponents for the Slater type orbitals must be entered by the user as part of the input file.

The WAVEPLOT tool offers the following plotting capabilities:

- Total charge density.
- Total spin polarisation.
- Difference between the total charge density and the density obtained by the superposition of the neutral atomic densities (visualisation of the charge shift).
- Electron density for individual levels.
- Real and imaginary part of the wavefunctions for individual levels.

7.1 Input for WAVEPLOT

The input file for WAVEPLOT must be named waveplot_in.hsd and should be a Human-friendly Structured Data (HSD) formatted file (see Appendix B) and must be present in the working directory.

The table below contains the list of the properties, which must occur in the input file waveplot_in.hsd:

Name	Type	Condition	Default	Page
Options	p		-	146
DetailedXML	s		-	
EigenvecBin	s		-	
GroundState	s		Yes	
Basis	p		-	149

Options Contains the options for WAVEPLOT. See p. 146.

DetailedXML Specifies the name of the file, which contains the detailed XML output of the DFTB+ calculation (presumably detailed.xml).

EigenvecBin Specifies the name of the file, which contains the eigenvectors in binary format (presumably eigenvec.bin).

GroundState Read ground or excited state occupation data from the detailed XML output.

Basis Contains the definition of the Slater-type orbitals which were used as basis in the DFTB+ calculation. At the moment, due to technical reasons this information has to be entered by the user per hand. In a later stage, it will be presumably read in by WAVEPLOT automatically. See p. 149.

Additionally optional definitions may also be present:

Name	Type	Condition	Default	Page
ParserOptions	p		{}	151

7.1.1 Options

This property contains the options (as a list of properties), which the user can set, in order to influence the behaviour of WAVEPLOT. The following properties can be specified:

PlottedRegion	p|m		-	148
NrOfPoints	3i		-	
PlottedKPoints	i+|m	periodic system	-	
PlottedLevels	i+|m		-	
PlottedSpins	i+|m		-	
TotalChargeDensity	l		No	
TotalSpinPolarisation	l		No	
TotalChargeDifference	l		No	
TotalAtomicDensity	l		No	
ChargeDensity	l		No	
RealComponent	l		No	
ImagComponent	l	complex wavefunction	No	
FoldAtomsToUnitCell	l	periodic system	No	
FillBoxWithAtoms	l		No	
NrOfCachedGrids	i		-1	
Verbose	l		No	
RepeatBox	3i		{1 1 1}	
ShiftGrid	l		Yes	
BinaryAccessTypes	2s		"stream" "stream"	

PlottedRegion Regulates the region which should be plotted. See p. 148.

NrOfPoints Specifies the resolution of the equidistant grid on which the various quantities should be calculated. The three integers represent the number of points along the three vectors of the parallelepiped specifying the plotted region. The number of all calculated grid points is the product of the three integers.

Example:

```
NrOfPoints = { 50 50 50 }     # 125 000 grid points
```

PlottedKPoints The list of integers specified here represent the k-points, at which the molecular orbitals should be plotted. The first k-point in the original DFTB+ calculation is represented by "1". The order of the specified k-points does not matter. (You can also use index specification expressions as described in appendix B.7.) The actual list of molecular orbitals to plot is obtained by intersecting the specifications for PlottedKPoints, PlottedLevels and PlottedSpins. The option is ignored if the original calculation was not periodic.

Example:

```
PlottedKPoints =  1 3 5     # The 1st, 3rd and 5th k-point is plotted
```

PlottedLevels The list of integers specified here represent the states which should be plotted. The first (lowest lying) state in the original DFTB+ calculation is represented by "1". The order of the specified states does not matter. (You can also use index specification expressions as described in appendix B.7.) The actual list of molecular orbitals to plot is obtained by intersecting the specifications for PlottedKPoints, PlottedLevels and PlottedSpins.

Example:

```
PlottedLevels = 1:-1     # All levels plotted
```

PlottedSpins The list of integers specified here represent the spins, for which the molecular orbitals should be plotted. The first spin in the original DFTB+ calculation is represented by "1". The order of the specified spins does not matter. (You can also use index specification expressions as described in appendix B.7.) The actual list of molecular orbitals to plot is obtained by intersecting the specifications for PlottedKPoints, PlottedLevels and PlottedSpins.

Example:

```
PlottedSpins =  1 2     # Both spin-up and spin-down plotted
```

ChargeDensity If true, the absolute square of the wavefunction is plotted for the selected molecular orbitals.

RealComponent If true, the real component of the wavefunction is plotted for the selected molecular orbitals.

ImagComponent If true, the imaginary component of the wavefunction is plotted for the selected molecular orbitals. This option is only parsed, if the wavefunctions in the DFTB+ calculation were complex.

TotalChargeDensity If true, the total charge density of the system is plotted.

TotalSpinPolarisation / TotalSpinPolarization If true, the total spin polarisation of the system (difference of the spin up and spin down densities) is plotted. This option is only interpreted if the processed DFTB+ calculation was spin polarised.

TotalChargeDifference If true, the difference between the total charge density and the charge density obtained by superposing the neutral atomic densities is plotted.

TotalAtomicDensity If true, the superposed neutral atomic densities are plotted.

FoldAtomsToUnitCell If true, the atoms are folded into the parallelepiped unit cell of the crystal.

FillBoxWithAtoms If true, the geometry is extended by those periodic images of the original atoms, which falls in the plotted region or on its borders. It sets FoldAtomsToUnitCell to Yes.

NrOfCachedGrids Specifies how many grids should be cached at the same time. The value -1 stands for as many as necessary to be as fast as possible. Since the plotted grids could eventually become quite big, you should set it to some positive non-zero value if you experience memory problems.

Example:

```
NrOfCachedGrids = 5     # Maximal 5 cached grids
```

RepeatBox The three integers specify how often the plotted region should be repeated in the generated cube files. Since repeating the grid is not connected with any extra calculations, this is a cheap way to visualise a big portion of a solid. You want probably set the FillBoxWithAtoms option to Yes to have the atoms also repeated (otherwise only the plotted function is repeated). In order to obtain the correct picture, you should set the plotted region to be an integer multiple of the unit cell of the crystal. Please note, that the phase of the wavefunctions in the repeated cells will be incorrect, except in the Γ-point.

Example:

```
RepeatBox = { 2 2 2 }     # Visualising a 2x2x2 supercell
```

ShiftGrid Whether the grid should be shifted, so that the specified origin lies in the middle of a cell and the grid fills out the specified plotted region symmetrically. The default is Yes. If set to No, the specified grid origin will be at the edge of a cell.

Verbose If true, some extra messages are printed out during the calculation.

BinaryAccessTypes Sets the file access type for binary I/O. See the keyword description in Sec. 2.5 on page 81.

PlottedRegion

Specifies the region, which should be included in the plot. You can specify it explicitly (as property list), or let WAVEPLOT specify it automatically using either the UnitCell{} or the OptimalCuboid{} methods.

Explicit specification Specifies origin and box size explicitly.

Origin	3r	-
Box	9r	-

Origin [*length*] Specifies the xyz coordinates of the origin as three real values.

Box [*length*] Specifies the three vectors which span the parallelepiped of the plotted region. The vectors are specified sequentially (a_{1x} a_{1y} a_{1z} a_{2x} a_{2y} a_{2z} a_{3x} a_{3y} a_{3z}). You are allowed to specify an arbitrary parallelepiped with nonzero volume here. Please note, however, that some visualisation tools only handles cube files with cuboid boxes correctly.

Example:

```
PlottedRegion = {
  Origin = { 0.0 0.0 0.0 }
  Box [Angstrom] = {
    12.5   12.5  -12.5
    12.5  -12.5   12.5
   -12.5   12.5   12.5
  }
}
```

UnitCell{} For the periodic geometries, this method specifies the plotted region to be spanned by the three lattice vectors of the crystal. The origin is set to (0 0 0). For cluster geometries, the smallest cuboid containing all atoms is constructed. For a cluster geometry the UnitCell{} object may have the following property:

MinEdgeLength	r	1.0

MinEdgeLength [*length*] Minimal side length of the cuboid, representing the plotted region. This helps to avoid cuboids with vanishing edge lengths (as it would be the case for a linear molecule).

Example:

```
PlottedRegion = UnitCell {
  MinEdgeLength [Bohr] = 2.0
}
```

OptimalCuboid{} Specifies the plotted region as a cuboid, which contains all the atoms and enough space around them, that no wavefunctions are leaking out of the cuboid. This object does not have any parameters.

Example:

```
PlottedRegion = OptimalCuboid {}
```

7.1.2 Basis

The basis definition is done by specifying the following properties:

Resolution	r	-	
ElementName1	p	-	151
ElementName2	p	-	151
⋮			

Resolution Specifies the grid distance used for discretising the radial wavefunctions. Setting it too small, causes a long initialisation time for WAVEPLOT. Setting it too high causes a very coarse grid with bad mapping and inaccurate charges. Values around 0.01 seem to work fine. (Units must be in Bohr.)

ElementName1 Basis for the first atom type. The name of this property is the name of that atom type.

ElementName2 Basis for the second atom type. The name of this property is the name of that atom type.

Before describing the properties (and their sub-properties) in detail, the full basis definition for carbon (sp) and hydrogen (s) should be presented as example:

```
Basis = {
  Resolution = 0.01
  C = {                          # Basis of the C atom
    AtomicNumber = 6
    Orbital = {                  # 2s orbital
      AngularMomentum = 0
      Occupation = 2
      Cutoff = 4.9
      Exponents = {  6.00000     3.00000     1.50000 }
      Coefficients = {
         1.050334389886e+01   2.215522018905e+01   9.629635264776e+00
        -4.827678012828e+01  -5.196013014531e+00  -2.748085126682e+01
         3.072783267234e+01  -1.007000163584e+01   8.295975876552e-01
      }
    }
    Orbital = {                  # 2p orbital
      AngularMomentum = 1
      Occupation = 2
      Cutoff = 5.0
      Exponents = {  6.00000     3.00000     1.50000  }
      Coefficients = {
        -2.659093036065e+00  -6.650979229497e+00  -1.092292307510e+01
         2.190230021657e+00  -9.376762008640e+00  -5.865448605778e-01
         8.208019468802e+00  -2.735743196612e+00   2.279582669709e-01
      }
    }
  }
  H = {                          # Basis for the H atom
    AtomicNumber = 1
    Orbital = {                  # 1s orbital
      AngularMomentum = 0
      Occupation = 1
      Cutoff = 4.2
      Exponents = {  2.00000     1.00000 }
      Coefficients = {
         1.374518455977e+01   1.151435212242e+01   2.072671588012e+00
        -1.059020844305e+01   3.160957468828e+00  -2.382382105798e-01
      }
    }
  }
}
```

Basis for an atom type

The actual basis for every atom type is specified as a property with the name of that type:

AtomicNumber	i	-	
Orbital	p	-	151
⋮			

AtomicNumber The atomic number of the species. This is not needed in the actual calculations, but for creating proper cube-files.

Orbital Contains the parameters of the orbitals. For every orbital a separate Orbital block must be created. See below.

Orbital For every orbital there is an orbital block which specifies the radial wavefunction. Thereby the following properties must be used:

AngularMomentum	i	-
Occupation	r	-
Cutoff	r	-
Exponents	r+	-
Coefficients	r+	-

AngularMomentum Angular momentum of the current orbital. ($s-0$, $p-1$, $d-2$, $f-3$)

Occupation Occupation of the orbital in the neutral ground state. (Needed to obtain the right superposed atomic densities.)

Cutoff Cutoff for the wave function. You should choose a value, where the value of $4\pi r^2 |R(r)|^2$ drops below 10^{-4} or 10^{-5}. $R(r)$ is the radial part of the wave function. If you do not have the possibility to visualise the radial wave function, take the half of the longest distance, for which an overlap interaction exists in the appropriate homonuclear Slater-Koster file. (Value must be entered in Bohr.)

Exponents The radial wave function with angular momentum l has the form:

$$R_l(r) = \sum_{i=1}^{n_{\text{exp}}} \sum_{j=1}^{n_{\text{pow}}} c_{ij}\, r^{l+j-1} e^{-\alpha_i r} \tag{7.1}$$

This property defines the multiplication factors in the exponent (α_i).

Coefficients This property contains the coefficients c_{ij} as defined in equation (7.1). The sequence of the coefficients must be as follows:

$c_{11}\ c_{12}\ \ldots\ c_{1n_{\text{pow}}}\ c_{21}\ c_{22}\ \ldots\ c_{2n_{\text{pow}}}\ \ldots$

7.1.3 ParserOptions

This block contains the options, which are effecting only the behaviour of the HSD parser and are not passed to the main program.

IgnoreUnprocessedNodes	l	No
StopAfterParsing	l	No

IgnoreUnprocessedNodes By default the code stops if it detects unused or erroneous keywords in the input. This *dangerous* flag suspends these checks. Use only for debugging purposes.

StopAfterParsing If set to Yes, the parser stops after processing the input and written out the processed input to the disc. It can be used to make sanity checks on the input without starting an actual calculation.

Chapter 8

SETUPGEOM

The program utility SETUPGEOM can help in preparing the input geometry for transport calculations, following the rules specified in the Transport section. Starting from a geometry that can be the output of a previous relaxation step or any other building step, SETUPGEOM can be used to specify the system partitioning into *contacts* and *device* regions and reorder the atom numbers such that the *device* is placed before the *contacts*. Additionally the tool reorders the atoms of the two PLs of each contact or can create the second PL if only one is specified. Finally, the *device* region is reordered and partitioned into PLs for more efficient Green's function calculations. A practical example is discussed in DFTB+ recipes.

8.1 Input for SETUPGEOM

The input of the code must be named `setup_in.hsd` and should be a Human-friendly Structured Data (HSD) formatted file (see Appendix B).

The file is similar to the DFTB+ input, where just 2 sections are needed. The table below contains the list of the properties, that must occur in the input file:

Name	Type	Condition	Default	Page
Geometry	plm		-	10
Transport	p		{}	

Geometry Specifies the geometry for the system to be calculated. See p. 10.

8.1.1 Transport{}

The transport block must specify the atoms in each contact. An example of the Transport section is reported below:

```
Transport {
  Contact {
   Id = "source"
    Atoms [zeroBased] = {9:24 56:78}
    ContactVector = 0.0 0.0 3.78
    PLsDefined = 2
  }
  Contact {
    Id = "drain"
    Atoms [zeroBased] = {81:100}
    ContactVector = 0.0 0.0 3.78
    PLsDefined = 2
  }
  Task = SetupGeometry{
    SlaterKosterFiles = type2names{
      ...
    }
    TruncateSKRange = {
      SKMaxDistance [AA] = 5.0
      HardCutOff = Yes
    }
  }
}
```

Id	s		
Atoms	l i		-
PLsDefined	i	1 or 2	2
ContactVector	3r		-
TruncateSKRange	p		
SlaterKosterFiles	p		

Id Specifies a unique contact name.

Atoms Sets the list of atoms belonging to the named contact. This list can be easily obtained using some external atomic manipulation tool like for instance Jmol.
NOTE the modifier [*zeroBased*] specifying that the defined atom numbers starts from 0 rather than 1. Use [*oneBased*] or no modifier for normal numbering starting from 1. (You can also use index specification expressions as described in appendix B.7.)

PLsDefined Specifies the number of PLs given for the named contact. If this value is 2 (default) the total number of atoms in the contact are divided by 2 and the 2nd PL is reordered according to the 1st with the help of ContactVector. If this value is 1, the correct numbers of PLs are created according to the interaction cutoff distance.

ContactVector Sets the translation vector connecting the 2 PLs along the transport direction. Since contact must be aligned to a cartesian axis, so must be this vector. Different contact can be in different directions. Also notice that the vector must be specified along the positive axis direction.

TruncateSKRange This section is the same as that described in section 2.4.

SlaterKosterFiles This section is the same as that described in section 2.4. The SK files are used to compute the cutoff distance.

8.1.2 Code output

The code writes two files, processed.gen and transport.hsd. The first file is the processed geometry, reordered according to the needs of transport calculations. NOTE that coordinates are folded to the unit cell such that all fractional coordinates are in the range 0 to 1. The structure is first translated such that all absolute coordinates have posive values. This step is important in order to take properly into account the periodic images. The file transport.hsd contains the details of the geometry partitioning for transport, as described in the Transport section and that can be included in the input file. For convenience this file also contains the block TruncateSKRange in order to make the Hamiltonian consistent with the MaxSKCutoff set in there.

Chapter 9

DFTB+ API

You can compile DFTB+ into a library and access some of its functionality via an API. Currently the API offers high level access only: you can set the current geometry and extract energy and forces for that geometry.

9.1 Building the library

In order to compile the DFTB+ library with the public API, set the WITH_API option to TRUE in the config.cmake configuration file. Then (from a separate build folder) build and install the code as ususal

```
cmake /PATH/TO/DFTBPLUS/SOURCE/FOLDER
make -j
```

After compilation, you can test the api functionality specifically with

```
ctest -R 'api_'
```

Finally, you can install the library with the usual install command

```
make install
```

After the installation, the library (libdftbplus.a) can be found in the lib/ folder of the installation directory. Depending on the build options this folder may contain several other libraries as well, which must be linked additional to libdftbplus.a to your binary.

9.2 General guidelines

Although the DFTB+ library contains nearly all internal routines of the DFTB+ code, you should access the code functionality only via the provided API and not by calling internal routines directly. We aim to keep the API stable over time, but the internal routines themselves can change without notice between releases. The API version can be found in the API_VERSION file in the src/dftbp/api/mm folder. We use semantic versioning, a change in the major (first) version number indicates backwards incompatible changes, while changes in the minor (second) version number indicate backwards compatible extensions of the API.

When using the API, we suggest that ParserVersion should be set in order to ensure that you can maintain backwards compatibility with later versions of DFTB+.

The Fortran interface is documented in the source code file src/dftbp/api/mm/mmapi.F90, while src/dftbp/api/mm/capi.F90 gives the bindings for calling from C. DFTB+ uses atomic units internally, hence exchanged values should be in this unit system (however HSD formatted data can carry unit modifiers, see examples of input parsing for details).

Appendix A

DFTB+ Releases

The recent input and parser versions along with the relevant code release date is listed below:

InputVersion / Release	ParserVersion	Date available
24.1	14	February 12, 2024
23.1	13	June 23, 2023
22.2	12	December 20, 2022
22.1	11	May 25, 2022
21.2	10	December 13, 2021
21.1	9	May 12, 2021
20.2	9	November 17, 2020
20.1	8	July 22, 2020
19.1	7	July 1, 2019
18.2	6	August 19, 2018
18.1	5	March 2, 2018
17.1	5	June 16, 2017

Appendix B

The HSD format

The Human-friendly Structured Data (HSD) format is a structured input format, which can be bijectively mapped onto a subset of the XML-language. Its simplified structure and notation should make it a more convenient user interface than reading and writing XML tags. This section contains a brief overview of the most important aspects of this format.

An input file in the HSD format consists basically of property assignments of the form

```
Property = value
```

where the value value was assigned to the property Property. The value must be one of the following types (detailed description of each follows later on):

- Scalar, such as
  - integer
  - real
  - complex
  - logical
  - string
- list of scalars
- method
- list of further property assignments

An unquoted hash mark (#) is interpreted as the start of a comment, everything after it, up to the end of the current line, is ignored by the parser (hash marks inside of quotes are taken as literals not comments):

```
# Entire line with comment
Prop1 = "hell#oo"  # Note, that the first hashmark is quoted!
```

The name of the properties, the methods and the logical values are case insensitive, so the assignments

```
Prop1 = 12
prOP1 = 12
Prop2 = Yes
Prop2 = YES
```

are pairwise identical. Quoted strings (specified either as a value for a property or as a file name), however, are case sensitive.

If a property, which should only appear once, is defined more than once, the parser uses the *first* definition and ignores all the other occurrences. *Thus specifying a property in the input a second time, does not override the first definition.* (For advanced use the HSD syntax also offers the possibility of conditional overriding/extending of previous definitions. For more details see B.6.)

B.1 Scalars and list of scalars

The following examples demonstrate the assignments with scalar types:

```
SomeInt = 1
SomeInt2 = -3
SomeRealFixedForm = 3.453
SomeRealExpForm = 2.12e-45
ComplexList = 4+9i  0.32-0.45i  3.2e-1+4.5e-1i  -1i  8.0
Logical1 = Yes
Logical2 = no
SomeString = "this is a string value"
```

As shown above, real numbers can be entered in either fixed or exponential form. Complex values are entered as two consecutive real valued numbers, one for the real part and the other for the imaginary component. The imaginary part must have a suffix of the letter i. No spaces are allowed inside the complex number, and the sign of the imaginary part must be explicitly written out (if the real part is present). If one of the two components is zero, it can be omitted.

The value for logical properties can be either Yes or No (and are case insensitive). Strings should always be enclosed in quotation marks, to make sure that they are treated as one string and that they are not interpreted by the parser:

```
String1 = "quoted string"
String2 = this value is actually a list of 9 strings  # list of strings!
String3 = "Method { ;"     # This is a string assignment
String4 = Method {         # This is syntactically incorrect, since
                       # it tries to assign a method to String4
```

A list of scalars is created by sequentially writing the scalars separated by one or more spaces:

```
PlottedLevels =  1 2 3
Origin =  0.0 0.0 0.0
ConfirmItTwice =  Yes Yes
SpeciesNames =  "Ga" "As"
```

The assignments statements are usually terminated by the end of the line. If the list of the assigned values goes over several lines, it must be entered between curly (brace) brackets. In that case, instead of the line end, the closing bracket will signal the end of the assignment. It is allowed to put a list of scalars in curly brackets, even if it is only one line long.

```
PlottedLevels = {
  1 2 3
}
Origin = { 0.0
0.0 0.0 }
Short = { 1 2 3 }
```

If you want to put more than one assignment in a line, you have to separate them with a semi-colon:

```
Variable = 12; Variable2 = 3.0
```

If a property should be defined as empty, either the empty list must be assigned to it or it must be defined as an empty assignment terminated by a semi-colon:

```
EmptyProperty = {}
EmptyProperty2 = ;
```

Please note, that explicitly specifying a property to be empty is not the same as not having specified it at all. In the latter case, the parser substitutes the default value for that property (if there is a default for it), while in the first case it interprets the property to be empty. If a property without default value is not specified, the parser stops with an appropriate error message.

B.2 Methods and property lists

Besides the scalar values and the list of scalars, the right hand side of an assignment may also contain a method, which itself may contain one or more scalar values or further property assignments as parameters:

```
Diagonaliser = LapackDAC {}     # Method without further params
PlottedLevels = Range { 1 3 }  # Range needs two scalar params
PlottedRegion = UnitCell {     # UnitCell needs a property list
  MinEdgeLength = 1.0          # as parameter
  SomeOtherProperty = Yes
}
```

The first assignment above is an example, where the method on the right hand side does not need any parameters specified. Please note, that even if no parameters are required, the opening and closing brackets after the method are mandatory. If the brackets are missing, the parser interprets the value as a string.

In the second assignment, the method Range needs only two integers as parameters, while for the method UnitCell several properties must be specified. A method may contain either nothing or scalars or property assignments, but never scalars and property assignments together. So the following assignment would be invalid:

```
InvalidSpecif = SomeMethod {
  1 2 3
  Property1 = 12
  "Some strings here"
}
```

Very often a value for the property is represented by a list of further property assignments (as above, but without naming an explicit method beforehand). In that case, the property assignments must be put between curly brackets (property list):

```
Options = {
  SubOption1 = 12
  Suboption2 = "string"
}
```

B.3 Modifiers

Each property may carry a modifier, which changes the interpretation of the assigned value:

```
LatticeConstant [Angstrom] = 12.23
```

Here, the property `LatticeConstant` possesses the `Angstrom` modifier, so the specified value will be interpreted to be in Ångström instead of the default length unit. Specifying a modifier for a property which is not allowed to carry one leads to parsing error.

The syntax of the HSD format also allows methods (used as values on the right hand side of an assignment) to carry modifiers, but this is usually not used in the current input structures.

Sometimes, the assigned value to a property contains several values with different units, so that more than one modifiers can be specified. In that case, the modifiers must be separated by a comma.

```
VolumeAndChargePerElement [Angstrom^3,au] = {
   1.2   0.3   # first element
   4.2   0.1   # second element
}
```

You have to specify either no modifier or all modifiers. If you want specify the default units for some of the quantities, you can omit the name of the appropriate modifier, but you must include the separating comma:

```
# Specifying the default unit for the charge. Note the separating comma!
VolumeAndChargePerElement [Angstrom^3,] = {
   1.2   0.3   # first element
   4.2   0.1   # second element
}
```

Specifying not enough or too many modifiers leads to parser error.

B.4 File inclusion

It is possible to include files in an HSD-formatted input by using the <<< and <<+ operators. The former includes the specified file as raw text without parsing it, while latter parses the included text:

```
Geometry = GenFormat {
  <<< "geo_start.gen"
}
Basis = {
  <<+ "File_containing_the_property_definitons_for_the_basis"
}
```

The file included with the <<+ operator must be a valid HSD document in itself.

B.5 Processing

After having parsed and processed the input file, the parser writes out the processed input to a separate file in HSD format. This file contains the internal representation for all properties, which can be specified by the user. In particular, all default values are explicitly set and all automatic definitions (e.g. ranges) are converted to their internal representations.

Assuming the following example as input

```
# Lattice contant specified in Angstrom.
# Internal representation uses Bohr, so it will be converted.
LatticeConstant [Angstrom] = 12.0

# This property is not set, as its commented out, so the
# default value will be set for this (let's assume, it's Yes)
#DoAProperJob = No

# Plotted levels specified as a range with parameter 1:3.
# This will be replaced by an explicit listing of the levels
PlottedLevels = { 1:3 }
```

the parsed and processed input (written to a special file) should look something like

```
LatticeConstant = 22.676713499923075
DoAProperJob = Yes
PlottedLevels = {
  1 2 3
}
```

If you want to reproduce your calculation later, you should use this processed input. It should give you identical results, even if the default setting for some properties had been changed in the code.

B.6 Extended format

As stated earlier, if a property, which should be defined only once, occurs more than once in the input, the parser uses per default the first definition and ignores all the others. Sometimes this is not the desired behaviour, therefore, the HSD format also offers the possibility to override properties that were set earlier. This feature can be very useful for scripts which are generate HSD input based on some user provided template. By just appending a few lines to the end of the user provided input the scripts can make sure that certain properties are set correctly. Thus, the script can modify the user input, without having to parse it at all.

The parser builds internally an XML DOM-tree from the HSD input. For every property or method name an XML tag with the same name (but lowercased) is created, which will contain the value of the property or the method. If the value contains further properties or methods, new XML tags are created inside the original one. Shortly, the HSD input is mapped on a tree, whereas the assignment and the containment (equal sign and curly brace) are turned into a parent-child relationships.[1] As an example an HSD input and the corresponding XML-representation is given below:

```
Level0Elem1 = 1                          <level0elem1>1</level0elem1>
Level0Elem2 = { 1 2 3 }                  <level0elem2>1 2 3</level0elem2>
Level0Elem3 = {                          <level0elem3>
  Level1Elem1 = 12                         <level1elem1>12</level1elem1>
  Level1Elem2 = Level2Elem1 {              <level1elem2>
                                             <level2elem1>
    Level3Elem1 = "abcd"                       <level3elem1>"abcd"</level3elem1>
    Level3Elem2 = {                            <level3elem2>
      Level4Elem1 = 12                           <level4elem1>12</level4elem1>
    }                                          </level3elem2>
  }                                          </level2elem1>
}                                          </level1elem2>
                                         </level0elem3>
```

By prefixing property and method names, the default behaviour of the parser can be overridden. Instead of creating a new tag (on the current encapsulation level) with the appropriate name, it will look for the *first occurrence* of the given tag and will process that one. Depending of the prefix character, the tag is processed in the following ways:

+: If the tag exists already, it's value is modified, otherwise the parser stops.

?: If the tag exists already, it's value is modified, otherwise the parser ignores the prefixed HSD construct.

***:** If the tag exists already, it's value is modified, otherwise it is created (and then it's value is modified).

/: If the tag does not exist yet, it is created and modified, otherwise the prefixed HSD construct is ignored.

!: The tag is newly created and modified. If it exists already, the old occurrence is deleted first.

The way the value of the tag is going to be modified, is ruled by the constructs inside the prefixed property or method name. If the parser finds non prefixed constructs here, the appropriate tags are

[1] In the internal tree representation of the HSD input there is no difference between properties and methods, both are just elements capable to contain some value or further elements. The differentiation in the HSD input is artificial and is only for human readability (equal sign after property names, curly brace after method names),

just added, otherwise the behaviour is determined by the rules above, just acting one level deeper in the tree. The following examples should make this a little bit more clear.

- Changing the value of Level0Elem1 to 3. If the element does not exist, it should be created with the value 3.

  ```
  !Level0Elem1 = 3
  ```

- Changing the value of Level0Elem3/Level1Elem1 to 21 (the slash indicates the parent-child relationship). If the element does not exist, stop with an error message:

  ```
  # Make sure the containing element exists. If yes, go inside, otherwise die.
  +Level0Elem3 = {
    # Set the value of Level1Elem1 or die, if it does not exist.
    +Level1Elem1 = 21
  }
  ```

  Please note, that each tag in the path must be prefixed. Using the following construct instead of the original one

  ```
  # Not prefixed, so it creates a new tag with empty value
  Level0Elem3 = {
    # The new tag doesn't contain anything, so the parser stops here
    +Level1Elem1 = 21
  }
  ```

  would end with an error message. Since Level0Elem1 is not prefixed here, a tag is created for it with an empty value (no children). It does not matter, whether the tag already existed before or not, a new tag is created and appended as the last element (last child) to the current block. Then the parser is processing its value. Due to the +Level1Elem1 directive it is looking for a child tag <level1elem1>. Since the tag was newly created, it does not contain any children, so the parser stops with an error message.

- Create a new tag Level1Elem3 inside Level0Elem3 with some special value. If the tag already exists, replace it.

  ```
  # Modifying the children of Level0Elem3 or dying if not present
  +Level0Elem3 = {
    # Replacing or if not existent creating Level1Elem3
    !Level1Elem3 = NewBlock {
      NewValue1 = 12
    }
  ```

  This example also shows, that the value for the new property can be any arbitrary complex HSD construct.

- Provide a default value "string" for Level0Elem3/Level1Elem2/Level2Elem1/Level3Elem1. If the tag is already present do not change its value.

```
# Modify Level0Elem3 or create it if non-existent
*Level0Elem3 = {
# Modify Level1Elem2 and Level2Elem1 or create them if non-existent
  *Level1Elem2 = *Level2Elem1 {
    # Create Level3Elem1 if non-existent with special value.
    /Level3Elem1 = "string"
  }
}
```

- If Level0Elem3/Level1Elem2 has the value Level2Elem1, make sure that Level3Elem1 in it exists, and has "" as value. If Level1Elem2 has a different value, do not change anything.

```
# If Level0Elem3 is present, process it, otherwise skip this block
?Level0Elem3 = {
  # The same for the next two containers
  ?Level1Elem2 = ?Level2Elem1 {
    # Create or replace Level3Elem1
    !Level3Elem1 = ""
  }
}
```

B.7 Index selection expressions

When specific elements should be selected from a list, the HSD parser allows for the usage of special index selection expressions. Typical examples are the selection of atoms (as in the MovedAtoms or Atoms keywords) or the selection of energy levels, k-points and spins (as in the WAVEPLOT keywords PlottedKPoints, PlottedSpins, PlottedLevels).

The expressions might contain the logical operators “not” (“!”), “and” (“&”) and “or” (“ ”, i.e. white space). The “not” operator has the highest and the “or” operator the lowest precedence. Parentheses can be used for grouping together operations. If atoms are being selected, atom type names can be used to select all atoms of a given type. The first element of the list has the index one. Negative indices are counted backwards from the last element of the list, with -1 corresponding to the last element. Index ranges can be specified as start:end (i.e. without white space as one word!), which inclusively selects all elements between start and end.

Below you find a few examples for selection expressions:

```
# Atoms 1, 2, 3, 4
MovedAtoms = 1 2 3 4 5

# Atoms within the range [1, 4] -> 1, 2, 3, 4
MovedAtoms = 1:4

# All atoms (from first to last)
MovedAtoms = 1:-1

# All atoms up to (and including) the fourth from the end
# (= all atoms apart the last 3)
MovedAtoms = 1:-4
```

```
# All Si atoms and atoms 2 and 4.
MovedAtoms = Si 2 4

# Any atoms from 2 to 6 which are Si atoms
MovedAtoms = 2:6 & Si

# All atoms, apart from atoms 2, 3, 4, 5, 6
MovedAtoms = !2:6

# All atoms from 2 to 6 which are not Si atoms
MovedAtoms = 2:6 & !Si

# All atoms apart from 1:3 and 5:7
MovedAtoms = !(1:3 5:7)

# All non-Si atoms in the ranges 1 to 10, 21 to 30 and 41 to 50.
MovedAtoms = !Si & (1:10 21:30 41:50)
```

The syntax of the index selection can be described by following formal grammar:

```
expr := addTerm { addTerm }
addTerm := mulTerm { "&" mulTerm }
mulTerm := term |  "!"term
term := selector | "("expr")"
selector := index{":"index} | speciesName
index := {-}[0-9]+
speciesName := [a-zA-Z][0-9a-zA-Z_]*
```

Appendix C

Unit modifiers

The DFTB+ code uses internally atomic units (with Hartree as the energy unit). The value of every numerical property in the input is interpreted to be in atomic units (au), unless the property carries a modifier.

The allowed modifiers and the corresponding conversion factors are given below.[1] (The modifiers are case insensitive). The symbol $\left[x^{-1}\right]$ indicates conversions, where not the value, but the reciprocal of the quantity is used in the conversion.

Ångströms may be listed as any of "a", "aa" or "angstrom" in the unit modifiers.

Length:	
Angstrom, AA, A (for Ångström)	0.188972598857892E+01
Meter, m	0.188972598857892E+11
pm	0.188972598857892E-01
Bohr, au	1.000000000000000E+00
Mass:	
amu	0.182288848492937E+04
au	1.000000000000000E+00
da	0.182288848492937E+04
dalton	0.182288848492937E+04
Volume:	
Angstrom^3, AA^3, A^3	0.674833303710415E+01
meter^3, m^3	0.674833303710415E+31
pm^3	0.674833303710415E-05
bohr^3, au	1.000000000000000E+00

[1]The conversion factors listed here were calculated with double precision on i686-linux architecture. Depending on your architecture, the values used there may deviate slightly.

Energy:	
Rydberg, Ry	0.500000000000000E+00
Electronvolt, eV	0.367493245336341E-01
kcal/mol	0.159466838598749E-02
Kelvin, K	0.316681534524639E-05
cm^-1	0.455633507361033E-05
Joule, J	0.229371256497309E+18
Hartree, Ha, au	1.000000000000000E+00
nm $\left[x^{-1}\right]$	0.45563355817434E+02
Force:	
eV/Angstrom, eV/AA, eV/A	0.194469064593167E-01
Joule/meter, J/m	0.121378050512919E+08
Hartree/Bohr, Ha/Bohr, au	1.000000000000000E+00
Time:	
femtosecond, fs	0.413413733365614E+02
picosecond, ps	0.413413733365614E+05
second, s	0.413413733365614E+17
au	1.000000000000000E+00
Charge:	
Coulomb, C	0.624150947960772E+19
au, e	1.000000000000000E+00
Velocity:	
au	1.000000000000000E+00
m/s	0.457102857516272E-06
pm/fs	0.457102857516272E-03
a/ps	0.457102857516272E-04
aa/ps	0.457102857516272E-04
angstrom/ps	0.457102857516272E-04
Pressure:	
Pa	0.339893208050290E-13
au	1.000000000000000E+00
Frequency:	
Hz	0.241888432650500E-16
THz	0.241888432650500E-04
cm^-1	0.725163330219952E-06
au	1.000000000000000E+00
Electric field strength:	
v/m	0.194469063788953E-11
au	1.000000000000000E+00
Dipole moment:	
CoulombMeter, Cm	0.117947426715764E+30
Debye	0.393430238326893E+00
au	1.000000000000000E+00

Angular units:

degrees, deg	1.745329251994330E-02
turns	6.283185307179586
gradians	1.570796326794897E-002
radians, rad	1.000000000000000E+00

Appendix D

Description of the gen format

The general (gen) format can be used to describe clusters, supercells and more exotic boundary conditions. It is based on the xyz format introduced with xmol, and extended to periodic and helical structures. Unlike some earlier implementations of gen, the format should not include any neighbour mapping information.

The first line of the file contains the number of atoms, n, followed by the type of geometry. C for cluster (non-periodic), S for supercell in Cartesian coordinates or F for supercell in fractions of the lattice vectors and H for one-dimensional periodic helical cells. The supercells are periodic in 3 dimensions, while helical cells repeat along the z direction (eventually) in addition to an optional rotational symmetry around that axis.

The second line contains the chemical symbols of the elements present separated by one or more spaces. The following n lines contain a list of the atoms. The first number is the atom number in the structure (this is currently ignored by the program). The second number is the chemical type from the list of symbols on line 2. Then follow the coordinates. For S and C format, these are x, y, z in Å, but for F they are fractions of the three lattice vectors.

If the structure is a supercell, the next line after the atomic coordinates contains a coordinate origin in Å. The last three lines are the supercell vectors in Å. For helical cells, there is a an origin line, and then an extra line containing three numbers (the repeat length of the cell along z in Å, the helical angle in degrees and the order of the C_n rotational symmetry around that axis). The symmetry of the resulting structure is the tensor product of the helical translation/twist axis and the C_n rotation operation. It is assumed that the screw axis is aligned parallel to z. These boundary condition lines are not present for cluster geometries.

Example: Geometry of GaAs with 2 atoms in the fractional supercell format

```
  2  F
# This is a comment
  Ga As
  1 1      0.0  0.0  0.0
  2 2      0.25 0.25 0.25
  0.000000      0.000000      0.000000
  2.713546      2.713546      0.
  0.            2.713546      2.713546
  2.713546      0.            2.713546
```

A single CH_2 chain deformed into a helix with a 1.25 Å cell height, twisting by 30° for each translation along *z* by this ammount.

```
  3  H
C  H
  1 1    1.016566615    1.777924612    0.000000000
  2 2    1.960988580    2.020972633    0.5133666820
  3 2    0.751006714    2.717254122   -0.5117698570
  0 0 0
  1.250   30.0 1
```

Note The DFTB+ input parser as well as the dptools utilities will ignore any lines starting with a # comment mark.

Appendix E

Atomic spin constants

These are suggested values for some atomic spin constants (W values) as given in reference [46], only the first two decimal places of the finite spin constants are numerically significant. These constants may eventually be included in the Slater-Koster files directly. Check the documentation of the Slater-Koster files required for a calculation to decide whether to use the LDA or PBE-GGA spin constants. Spin constants for range-separated functionals are given together with the respective Slater-Koster files at www.dftb.org.

W		LDA			PBE		
		s	p	d	s	p	d
H	s	-0.064			-0.072		
C	s	-0.028	-0.024		-0.031	-0.025	
	p	-0.024	-0.022		-0.025	-0.023	
N	s	-0.030	-0.026		-0.033	-0.027	
	p	-0.026	-0.025		-0.027	-0.026	
O	s	-0.032	-0.028		-0.035	-0.030	
	p	-0.028	-0.027		-0.030	-0.028	
Si	s	-0.018	-0.013	0.000	-0.020	-0.015	0.000
	p	-0.013	-0.012	0.000	-0.015	-0.014	0.000
	d	0.000	0.000	-0.019	0.002	0.002	-0.032
S	s	-0.019	-0.016	0.000	-0.021	-0.017	0.000
	p	-0.016	-0.014	0.000	-0.017	-0.016	0.000
	d	0.000	0.000	-0.010	0.000	0.000	-0.080
Fe	s	-0.013	-0.009	-0.003	-0.016	-0.012	-0.003
($3d^7 4s^1$)	p	-0.009	-0.011	-0.001	-0.012	-0.029	-0.001
	d	-0.003	-0.001	-0.015	-0.003	-0.001	-0.015
Ni	s	-0.009	-0.009	-0.003	-0.016	-0.012	-0.003
	p	-0.009	-0.010	-0.001	-0.012	-0.022	-0.001
	d	-0.003	-0.001	-0.017	-0.003	-0.001	-0.018

Appendix F

Slater-Kirkwood dispersion constants

The following table contains recommended dispersion constants for some elements with the Slater-Kirkwood dispersion model (see p. 58). The values have been tested for biological systems, C, N, O and H predominantly for DNA [22]. If you would like to calculate different systems or you're looking for other elements, check references [63] and [42]. The values of the atomic polarisabilities and cutoffs are given for zero, one, two, three, four and more than four neighbours.

Element	Polarisability [Å^3]	Cutoff [Å]	Chrg	Note
O	0.560 0.560 0.000 0.000 0.000 0.000	3.8 3.8 3.8 3.8 3.8 3.8	3.15	
N	1.030 1.030 1.090 1.090 1.090 1.090	3.8 3.8 3.8 3.8 3.8 3.8	2.82	
C	1.382 1.382 1.382 1.064 1.064 1.064	3.8 3.8 3.8 3.8 3.8 3.8	2.50	
H	0.386 0.386 0.000 0.000 0.000 0.000	3.5 3.5 3.5 3.5 3.5 3.5	0.80	
P	1.600 1.600 1.600 1.600 1.600 1.600	4.7 4.7 4.7 4.7 4.7 4.7	4.50	PO_4 only
S	3.000 3.000 3.000 3.000 3.000 3.000	4.7 4.7 4.7 4.7 4.7 4.7	4.80	S, not SO_2

Appendix G

DftD3 dispersion constants

These are suggested dispersion values for some of the DFTB parameterizations for use with the DftD3 model (see p. 59). The table below gives the old defaults for DFTB3, those for which the 3OB parameters were fitted at the halogen correction stage, along with choices for the various OB2 parameterizations for range separated calculations.

Becke-Johnson damping	old default	3OB	OB2 (base)	OB2 (shift)	OB2 (split)
a_1	0.5719	0.746	0.717	0.816	0.497
a_2	3.6017	4.191	2.565	2.057	3.622
s_6	1.0	1.0	1.0	1.0	1.0
s_8	0.5883	3.209	0.011	0.010	0.010

Note: for the H5 corrected model, the DftD3 zero-damping parameters given on page 76 should be used.

Appendix H

DftD4 dispersion constants

These are suggested dispersion values for some of the DFTB parameterizations for use with the D4 model (see p. 62). The table below gives parameterizations with and without non-additive dispersion included.

Table H.1: Becke–Johnson damping parameters for various Slater–Koster parametrizations of the DFTB hamiltonian. Parametrizations are done both with non-additive contributions and without.

parameters	s_6	s_8	s_9	a_1	a_2 $[a_0]$
3ob	1	0.4727337	0	0.5467502	4.4955068
	1	0.6635015	1	0.5523240	4.3537076
matsci	1	2.7711819	0	0.4681712	5.2918629
	1	3.3157614	1	0.4826330	5.3811976
mio	1	1.1948145	0	0.6074567	4.9336133
	1	1.2916225	1	0.5965326	4.8778602
ob2(base)	1	2.7611320	0	0.6037249	5.3900004
	1	2.9692689	1	0.6068916	5.4476789
pbc	1	1.7303734	0	0.5546548	4.7973454
	1	2.1667394	1	0.5646391	4.9576353

Appendix I

Solvent parameters

Table I.1: Names and associated constants for available solvents.

Name	Dielectric constant	Molar mass [g/mol]	Mass density [kg/L]
acetone	20.7	58.08	0.79
acetonitrile	37.5	41.05	0.786
toluene	7.0	92.14	0.867
benzene, c6h6	7.0	78.11	0.867
chloroform, trichloromethane, chcl3	7.0	119.38	1.49
dichloromethane, ch2cl2	7.0	84.93	1.33
cs2	2.6	76.13	1.266
dmf	37.0	73.1	0.95
dmso	47.2	78.13	1.1
ether	7.3	74.12	0.713
water, h2o	78.5	18.0	0.998
methanole	33.6	32.04	0.792
nhexane, n-hexane	1.88	86.18	0.66
thf	10.0	72.1061	0.883

Appendix J

Atomic onsite constants

These are suggested values for some atomic on-site correction constants as given in reference [26], see p. 78.

W		PBE		
		s	*p*	*d*
$H_{\uparrow\uparrow}$	*s*	0.000		
$H_{\uparrow\downarrow}$	*s*	0.000		
$C_{\uparrow\uparrow}$	*s*	0.00000	0.04973	
	p	0.04973	-0.01203	
$C_{\uparrow\downarrow}$	*s*	0.00000	0.10512	
	p	0.10512	0.02643	
$N_{\uparrow\uparrow}$	*s*	0.00000	0.06816	
	p	0.06816	-0.00879	
$N_{\uparrow\downarrow}$	*s*	0.00000	0.12770	
	p	0.12770	0.03246	
$O_{\uparrow\uparrow}$	*s*	0.00000	0.08672	
	p	0.08672	-0.00523	
$O_{\uparrow\downarrow}$	*s*	0.00000	0.14969	
	p	0.14969	0.03834	
$S_{\uparrow\uparrow}$	*s*	0.00000	0.07501	0.00398
	p	0.07501	0.00310	0.01100
	d	0.00398	0.01100	-0.01792
$S_{\uparrow\downarrow}$	*s*	0.00000	0.11653	0.03915
	p	0.11653	0.03058	0.04979
	d	0.03915	0.04979	0.01582
$Ti_{\uparrow\uparrow}$	*s*	0.00000	0.02659	-0.00587
	p	0.02659	-0.01297	-0.00523
	d	-0.00587	-0.00523	-0.00750
$Ti_{\uparrow\downarrow}$	*s*	0.00000	0.06881	0.01239
	p	0.06881	0.01640	0.01144
	d	0.01239	0.01144	0.02604
$Au_{\uparrow\uparrow}$	*s*	0.00000	0.03752	0.00073
	p	0.03752	-0.00505	-0.00002
	d	0.00073	-0.00002	0.00531
$Au_{\uparrow\downarrow}$	*s*	0.00000	0.06928	0.01339
	p	0.06928	0.01677	0.01228
	d	0.01339	0.01228	0.02519

Appendix K

Hartree Hubbard constants for pp-RPA calculations

These are suggested values for some Hubbard-like parameters employed in particle-particle Random Phase Approximation calculations (see p. 91):

Element	HHubbard
O	0.59637
C	0.49748
N	0.56235
H	0.68353

Appendix L

Description of restart files

L.0.1 charges.bin / charges.dat

Initial charges and the current orbital charges are stored in these files. Both contain the same information, but charges.bin is stored as unformatted binary data, while charges.dat is a text file.

The first line of the file is:

```
  version tBlockCharges tImaginaryBlock nSpin CheckSum
```

Where version is currently 3, tBlockCharges and tImaginaryBlock are logical variables as to whether real and imaginary block charges are present. nSpin is the number of spin channels (1, 2 or 4 for spin free, collinear and non-collinear) and checksum is the totals for the charges in each spin channel.

The subsequent nAtom $\times$ nSpin lines contain the individual orbital occupations for each atom in spin channel 1 (then 2 ... 4, if present).

If tBlockCharges is true, then the on-site block charges for each atom and spin channel are stored, followed by the imaginary part if tImaginaryBlock is true.

Examples of the contents of charges.dat are given below for an H_2O molecule in the *yz* aligned with its dipole along *y*. Using the mio-1-1 Slater-Koster set, this file would contain:

```
           3 F F           1   8.0000000000000018
  6.5926151655316767       0.0000000000000000       0.0000000000000000       0.0000000000000000
 0.70369241723366482
 0.70369241723466003
```

When OrbitalResolved = No. So, this is version 3 of the format, without block charges and it is spin free with 8 electrons. The electronic charges are grouped into the lowest atomic orbitals of each atom in this case. There is some small numerical noise in some of these these values ($< 10^{-14}$).

With OrbitalResolved = Yes, the oxygen has 1.7 2*s* electrons and 4.83 2*p* orbitals (electrons listed in the lowest labelled state in each shell).

```
           3 F F           1   8.0000000000000018
  1.7335403452609417       4.8346073382345685       0.0000000000000000       0.0000000000000000
 0.71592615825295036
 0.71592615825154060
```

While for a pseudo-SIC calculation, where the net spin is 0:

```
           3 T F           2   8.0000000000000018       0.0000000000000000
  1.7566193972978825       1.7147230821039328       1.2018994732683752       2.0000000000000013
 0.66337902366501833
 0.66337902366479040
```

```
  0.0000000000000000       0.0000000000000000       0.0000000000000000       0.0000000000000000
  0.0000000000000000
  0.0000000000000000
  1.7566193972978825      -0.28455491415632111      7.9592308124161928E-014  -7.7482799007142168E-027
 -0.28455491415632111      1.7147230821039328       5.6922736516833719E-014   4.1776281206242259E-026
  7.9592308124161928E-014   5.6922736516833719E-014   1.2018994732683752       -4.6749196043609904E-016
 -7.7482799007142168E-027   4.1776281206242259E-026  -4.6749196043609904E-016   2.0000000000000013
 0.66337902366501833
 0.66337902366479040
  0.0000000000000000       0.0000000000000000       0.0000000000000000       0.0000000000000000
  0.0000000000000000       0.0000000000000000       0.0000000000000000       0.0000000000000000
  0.0000000000000000       0.0000000000000000       0.0000000000000000       0.0000000000000000
  0.0000000000000000       0.0000000000000000       0.0000000000000000       0.0000000000000000
  0.0000000000000000
  0.0000000000000000
```

Note the 0 blocks for the spin polarisation in channel 2, and also that the diagonal of the block charges matches the orbital charges for the atoms.

L.0.2 contact.bin / contact.dat

Self-consistent transport calculations require contact potential shifts. The format of the shiftcont_* files are either ascii (.dat) or binary (.bin). The shiftcont_*.dat files have the following format:

The first two lines of the file are:

```
version
nContactAtoms maxShells spinChannels tBlockCharges
```

- version: The file format is currently version 1
- nContactAtoms: Number of atoms in the contact
- maxShells: Maximum number of angular shells on the contact atoms
- spinChannels: Number of spin channels in the system (1 for spin free, 2 for spin polarized)
- tBlockCharges: a logical flag (T/F) as to whether block charges are present in the file (these are required for +U calculations).

This is then followed by lines for

- The number of orbitals on the atoms
- Shifts for the shells of the atoms
- Charges for individual orbitals

If tBlockCharges is true, the block shifts and charges are then given for each spin channel and atom.

Finally the Fermi level(s) for the contact are printed (this can be over-ridden in the input at calculation time, see the FermiLevel keyword in section 4.2.2).

An earlier format for contacts is also supported. This lacks the first line containing the version number, along with the logical flag and sections relating to block charges.

L.0.3 tddump.bin / tddump.dat

The restart data for a real time propagation calculation is stored in either the native binary format of your computer (`tddump.bin`) or as ascii text (`tddump.dat`) format.

The content of the file given is below (all quantities are in atomic units). In the case of ascii output this is broken into lines:

- version: The file format, currently version 1
- nOrbitals nSpin$\times$nKpoints nAtoms current_simulation_time timestep_size
- nOrbitals^2 $\times$ nSpin $\times$ nKpoints lines of the (complex) density matrix at current time
- nOrbitals^2 $\times$ nSpin $\times$ nKpoints lines of the (complex) density matrix at the previous timestep
- nAtoms lines of cartesian coordinates of the atoms at current time
- nAtoms lines of atomic velocities at current time

Appendix M

Publications to cite

The following publications should be considered for citation, if you are publishing any results calculated by using DFTB+.

DFTB+ code	[37]
non-SCC DFTB	[77], [84]
SCC DFTB	[23]
Collinear spin polarisation	[45]
Non-collinear spin polarisation	[44]
Spin orbit coupling	[44]
QM/MM coupling (external charges)	[18], [33]
Van der Waals interaction (dispersion)	[22]
DFTB+U	[39]
3rd order corrections	[93]
ΔDFTB excited states	[48]
linear-response TD-DFTB	[67]
real-time TD-DFTB	[14]
REKS calculations	[54]
range-separated hybrid DFTB	[66], [57]
non-adiabatic coupling vectors	[68], [69]

Bibliography

[1] https://github.com/opencollab/arpack-ng. 87

[2] *Introducing Molecular Electronics*, chapter Tight-Binding DFT for Molecular Electronics (gDFTB), pages 153–184. Lecture Notes in Physics. Springer, 2005. 124

[3] Alberto Ambrosetti, Anthony M. Reilly, Robert A. DiStasio, Jr., and Alexandre Tkatchenko. Long-range correlation energy calculated from coupled atomic response functions. *J. Chem. Phys.*, 140:18A508, 2014. 57, 66

[4] H. C. Andersen. Molecular dynamics at constant pressure and/or temperature. *J. Chem. Phys.*, 72:2384, 1980. 23

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. 44

[6] B. Aradi, B. Hourahine, and Th. Frauenheim. DFTB+, a sparse matrix-based implementation of the DFTB method. *J. Phys. Chem. A*, 111(26):5678, 2007. 106

[7] B. Aradi, A. M. N. Niklasson, and T. Frauenheim. Extended lagrangian density functional tight-binding molecular dynamics for molecules and solids. *J. Chem. Theory Comput.*, 11:3357–3363, 2015. 26, 79

[8] Vilhjálmur Ásgeirsson, Christoph A Bauer, and Stefan Grimme. Quantum chemical calculation of electron ionization mass spectra for general organic and inorganic molecules. *Chem. Sci.*, 8(7):4879–4895, 2017. 36

[9] Christoph Bannwarth, Eike Caldeweyher, Sebastian Ehlert, Andreas Hansen, Philipp Pracht, Jakob Seibert, Spicher Spicher, and Stefan Grimme. Extended tight-binding quantum chemistry methods. *WIREs Comput. Mol. Sci.*, 11:e01493, 2020. 36

[10] Christoph Bannwarth, Sebastian Ehlert, and Stefan Grimme. Gfn2-xtb—an accurate and broadly parametrized self-consistent tight-binding quantum chemical method with multipole electrostatics and density-dependent dispersion contributions. *J. Chem. Theory Comput.*, 15(3):1652–1671, 2019. 36

[11] Michael J Bearpark, Michael A Robb, and H Bernhard Schlegel. A direct method for the location of the lowest energy point on a potential surface crossing. *Chem. Phys. Lett.*, 223(3):269–274, 1994. 90

[12] H. J. C. Berendsen, J. P. M. Postma, W. F. Van Gunsteren, A. Dinola, and J. R. Haak. Molecular-dynamics with coupling to an external bath. *J. Chem. Phys.*, 81(8):3684–3690, 1984. 24, 25, 26

[13] Erik Bitzek, Pekka Koskinen, Franz Gähler, Michael Moseler, and Peter Gumbsch. Structural relaxation made simple. *Phys. Rev. Lett.*, 97:170201, Oct 2006. 15, 20

[14] Franco P. Bonafé, Bálint Aradi, Ben Hourahine, Carlos R. Medrano, Federico J. Hernández, Thomas Frauenheim, and Cristián G. Sánchez. A Real-Time Time-Dependent Density Functional Tight-Binding Implementation for Semiclassical Excited State Electron-Nuclear Dynamics and Pump-Probe Spectroscopy Simulations. *Journal of Chemical Theory and Computation*, jun 2020. 92, 195

[15] Eike Caldeweyher, Christoph Bannwarth, and Stefan Grimme. Extension of the D3 dispersion coefficient model. *J. Chem. Phys.*, 147(3):034112, 2017. 57

[16] Eike Caldeweyher, Sebastian Ehlert, Andreas Hansen, Hagen Neugebauer, Sebastian Spicher, Christoph Bannwarth, and Stefan Grimme. A generally applicable atomic-charge dependent London dispersion correction. *J. Chem. Phys.*, 150(15):154122, 2019. 57, 62, 64

[17] M. Ceriotti, J. More, and D. E. Manolopoulos. i-pi: A python interface for ab initio path integral molecular dynamics simulations. *Computer Phys. Comm.*, 185:1019–1026, 2014. 29

[18] Q. Cui, M. Elstner, T. Frauenheim, E. Kaxiras, and M. Karplus. Combined self-consistent charge density functional tight-binding (SCC-DFTB) and CHARMM. *J. Phys. Chem. B*, 105:569, 2001. 195

[19] A. Domínguez, B. Aradi, T. Frauenheim, V. Lutsker, and T. A. Niehaus. Extensions of the time-dependent density functional based tight-binding approach. *Journal of Chemical Theory and Computation*, 9(11):4901–4914, 2013. 90

[20] A. Domínguez, T. A. Niehaus, and T. Frauenheim. Accurate hydrogen bond energies within the density functional tight binding method. *The Journal of Physical Chemistry A*, 119(14):3535–3544, 2015. 34, 78

[21] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. Accelerating numerical dense linear algebra calculations with gpus. *Numerical Computations with GPUs*, pages 1–26, 2014. 44

[22] M. Elstner, P. Hobza, T. Frauenheim, S. Suhai, and E. Kaxiras. Hydrogen bonding and stacking interactions of nucleic acid base pairs: a density-functional-theory based treatment. *J. Chem. Phys.*, 114:5149, 2001. 57, 58, 59, 179, 195

[23] M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, T. Frauenheim, S. Suhai, and G. Seifert. Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties. *Phys. Rev. B*, 58:7260, 1998. 79, 195

[24] V. Eyert. A comparative study on methods for convergence acceleration of iterative vector sequences. *J. Comp. Phys.*, 124:271, 1996. 39

[25] T. Frauenheim, G. Seifert, M. Elstner, T. Niehaus, C. Kohler, M. Amkreutz, M. Sternberg, Z. Hajnal, A. Di Carlo, and S. Suhai. Atomistic simulations of complex materials: ground-state and excited-state properties. *J. Phys. Condens. Matter*, 14(11):3015–3047, Mar 2002. 7

[26] Adriel Dominguez Garcia. *Density functional approaches for the interaction of metal oxides with small molecules.* PhD thesis, Universität Bremen, 2014. http://elib.suub.uni-bremen.de/edocs/00103868-1.pdf. 78, 109, 187

[27] M. Gaus, Q. Cui, and M. Elstner. DFTB3: Extension of the Self-Consistent-Charge Density-Functional Tight-Binding Method (SCC-DFTB). *J. Chem. Theory Comput.*, 7:931–948, 2011. 66, 67, 75

[28] Nir Goldman, Bálint Aradi, Rebecca K. Lindsey, and Laurence E. Fried. Development of a multicenter density functional tight binding model for plutonium surface hydriding. *J. Chem. Theory Comput.*, 14:2652–2660, 2018. 33

[29] S. Grimme, J. Antony, S. Ehrlich, and H. Krieg. A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu. *J. Chem. Phys.*, 132:154104, 2010. 57, 60, 65

[30] S. Grimme, S. Ehrlich, and L. Goerigk. Effect of the damping function in dispersion corrected density functional theory. *J. Chem. Phys.*, 32:1456–1465, 2011. 57, 60

[31] Stefan Grimme, Christoph Bannwarth, and Philip Shushkov. A robust and accurate tight-binding quantum chemical method for structures, vibrational frequencies, and noncovalent interactions of large molecular systems parametrized for all spd-block elements (Z=1–86). *J. Chem. Theory Comput.*, 13(5):1989–2009, 2017. 36

[32] M. J. Han, T. Ozaki, and J. Yu. O(N) LDA+U electronic structure calculation method based on the nonorthogonal pseudoatomic orbital basis. *Phys. Rev. B*, 73:045110, 2006. 44

[33] W. Han, M. Elstner, K. J. Jalkanen, T. Frauenheim, and S. Suhai. Hybrid SCC-DFTB/molecular mechanical studies of H-bonded systems and of N-acetyl-(L-Ala)$_n$-N'-Methylamide helices in water solution. *Int. J. Quant. Chem.*, 78:459, 2000. 195

[34] Yu Harabuchi, Miho Hatanaka, and Satoshi Maeda. Exploring approximate geometries of minimum energy conical intersections by TDDFT calculations. *Chemical Physics Letters: X*, 2:100007, 2019. 90

[35] S. C. Harvey, R. K. Z. Tan, and T. E. Cheatham. The flying ice cube: Velocity rescaling in molecular dynamics leads to violation of energy equipartition. *J. Comput. Chem.*, 19(7):726–740, 1998. 24

[36] D. Heringer, T. A. Niehaus, M. Wanko, and T. Frauenheim. Analytical excited state forces for the time-dependent density-functional tight-binding method. *J. Comput. Chem.*, 28(16):2589, 2007. 89

[37] B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, and T. Frauenheim. DFTB+, a software package for efficient approximate density functional theory based atomistic simulations. *J. Chem. Phys.*, 152(12):124101, 2020. 7, 195

[38] B Hourahine, B Aradi, and T Frauenheim. DFTB+ and lanthanides. *J. Phys: Conf. Ser.*, 242(1):012005, jul 2010. 37

[39] B. Hourahine, S. Sanna, B. Aradi, C. Köhler, T. Niehaus, and Th. Frauenheim. Self-interaction and strong correlation in DFTB. *J. Phys. Chem. A*, 111(26):5671, 2007. 54, 195

[40] Wonpil Im, Michael S. Lee, and Charles L. Brooks III. Generalized Born model with a simple smoothing function. *J. Comput. Chem.*, 24(14):1691–1702, 2003. 73

[41] D. D. Johnson. Modified Broyden's method for accelerating convergence in self consistent calculations. *Phys. Rev. B*, 38:12807, 2003. 38

[42] Y. K. Kang and M. S. Jhon. Additivity of atomic static polarizabilities and dispersion coefficients. *Theoretica Chimica Acta*, 61:41, 1982. 179

[43] Andreas Klamt and G Schüürmann. COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient. *Journal of the Chemical Society, Perkin Transactions 2*, (5):799–805, 1993. 71

[44] C. Köhler, T. Frauenheim, B. Hourahine, G. Seifert, and M. Sternberg. Treatment of collinear and noncollinear electron spin within an approximate density functional based method. *J. Phys. Chem. A*, 111(26):5622, 2007. 195

[45] C. Köhler, G. Seifert, and T. Frauenheim. Density-functional based calculations for Fe(n), ($n \leq 32$). *Chem. Phys.*, 309:23, 2005. 195

[46] Christof Köhler. *Berücksichtigung von Spinpolarisationseffekten in einem dichtefunktional-basierten Ansatz*. PhD thesis, Department Physik der Fakultät fur Naturwissenschaften an der Universität Paderborn, 2004. http://ubdata.uni-paderborn.de/ediss/06/2004/koehler/. 177

[47] A. Kovalenko, S. Ten-no, and F. Hirata. Solution of three-dimensional reference interaction site model and hypernetted chain equations for simple point charge water by modified method of direct inversion in iterative subspace. *J. Comput. Chem.*, 20:928–936, 1999. 20

[48] T. Kowalczyk, K. Le, and S. Irle. Self-consistent optimization of excited states within density-functional tight-binding. *J. Chem. Theory Comput.*, 12:313–323, 2016. 49, 195

[49] Julian J. Kranz, Marcus Elstner, Bálint Aradi, Thomas Frauenheim, Vitalij Lutsker, Adriel Dominguez Garcia, and Thomas A. Niehaus. Time-dependent extension of the long-range corrected density functional based tight-binding method. *J. Chem. Theory Comput.*, 13(4):1737–1747, 2017. 90

[50] M. Kubillus, T. Kubař, M. Gaus, Jan Řezáč, and M. Elstner. Parameterization of the DFTB3 method for Br, Ca, Cl, F, I, K, and Na in organic and biological systems. *J. Chem. Theory Comput.*, 11:332–342, 2014. 75

[51] Adrian W. Lange and John M. Herbert. Improving generalized Born models by exploiting connections to polarizable continuum models. i. an improved effective coulomb operator. *J. Chem. Theory Comput.*, 8(6):1999–2011, 2012. PMID: 26593834. 69

[52] Stephan Lany and Alex Zunger. Accurate prediction of defect properties in density functional supercell calculations. *Modelling and Simulation in Materials Science and Engineering*, 17(8):084002, Nov 2009. 104

[53] V. I. Lebedev and D. N. Laikov. A quadrature formula for the sphere of the 131st algebraic order of accuracy. *Doklady Mathematics*, 59:477–481, 1999. 72, 74

[54] I. S. Lee, M. Filatov, and S. K. Min. Formulation and implementation of the spin-restricted ensemble-referenced kohn-sham method in the context of the density functional tight binding approach. *J. Chem. Theory Comput.*, 15(5):3021–3032, 2019. 97, 98, 195

[55] R. B. Lehoucq, D. C. Sorensen, and C. Yang. Arpack users guide: Solution of large scale eigenvalue problems by implicitly restarted arnoldi methods., 1997. 87, 90, 108

[56] Filippo Lipparini, Benjamin Stamm, Eric Cances, Yvon Maday, and Benedetta Mennucci. Fast domain decomposition algorithm for continuum solvation models: Energy and first derivatives. *J. Chem. Theory Comput.*, 9(8):3637–3648, 2013. 71

[57] V. Lutsker, B. Aradi, and T. A. Niehaus. Implementation and benchmark of a long-range corrected functional in the densityfunctional based tight-binding method. *J. Chem. Phys.*, 143:184107, 2015. 77, 195

[58] M Mantina, R Valero, CJ Cramer, and DG Truhlar. *CRC Handbook of Chemistry and Physics*. CRC Press Boca Raton, FL, 2010. 71

[59] Manjeera Mantina, Adam C. Chamberlin, Rosendo Valero, Christopher J. Cramer, and Donald G. Truhlar. Consistent van der waals radii for the whole main group. *J. Phys. Chem. A*, 113(19):5806–5812, 2009. 72

[60] Aleksandr V. Marenich, Steven V. Jerome, Christopher J. Cramer, and Donald G. Truhlar. Charge model 5: An extension of Hirshfeld population analysis for the accurate description of molecular interactions in gaseous and condensed phases. *J. Chem. Theory Comput.*, 8(2):527–541, 2012. 71, 82

[61] G. J. Martyna, M. E. Tuckerman, D. J. Tobias, and M. L. Klein. Explicit reversible integrators for extended systems dynamics. *Molecular Phys.*, 87:1117–1157, 1996. 24, 25

[62] M. Methfessel and A. T. Paxton. High-precision sampling for Brillouin-zone integration in metals. *Phys. Rev. B*, 40:3616, 1989. 48

[63] K. J. Miller. Additivity methods in molecular polarizability. *J. Am. Chem. Soc.*, 112:8533, 1990. 179

[64] H. J. Monkhorst and J. D. Pack. Special points for Brillouin-zone integrations. *Phys. Rev. B*, 13:5188, 1976. 52

[65] H. J. Monkhorst and J. D. Pack. "special points for Brillouin-zone integrations"–a reply. *Phys. Rev. B*, 16:1748, 1977. 52

[66] T. A. Niehaus and F. Della Sala. Range separated functionalsin the density functional based tight-binding method: Formalism. *Phys. Status Solidi B*, 249(2):237–244, 2012. 77, 195

[67] T. A. Niehaus, S. Suhai, F. Della Sala, P Lugli, M. Elstner, G. Seifert, and Th. Frauenheim. Tight-binding approach to time-dependent density-functional response theory. *Phys. Rev. B*, 63:085108, 2001. 87, 195

[68] Thomas A Niehaus. Ground-to-excited derivative couplings for the density functional-based tight-binding method: semi-local and long-range corrected formulations. *Theoretical Chemistry Accounts*, 140(4):34, 2021. 89, 195

[69] Thomas A Niehaus. Exact non-adiabatic coupling vectors for the time-dependent density functional based tight-binding method. *The Journal of Chemical Physics*, 158:054103, 2023. 89, 195

[70] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006. 20

[71] Alexey Onufriev, Donald Bashford, and David A. Case. Exploring protein native states and large-scale conformational changes with a modified generalized born model. *Proteins*, 55(2):383–394, 2004. 69

[72] Alexey V. Onufriev and David A. Case. Generalized Born implicit solvent models for biomolecules. *Annu. Rev. Biophys.*, 48(1):275–296, 2019. 67

[73] A. Pecchia and A. Di Carlo. Atomistic theory of transport in organic and inorganic nanostructures. *Rep. Prog. Phys.*, 67:1497, 2004. 124

[74] A. Pecchia, G. G Penazzi, L. Salvucci, and A. Di Carlo. Non-equilibrium Green's functions in density functional tight binding: method and applications. *New J. of Physics*, 10(6):065022, 2008. 118, 124

[75] A. G. Petukhov, I. I. Mazin, L. Chioncel, and A. I. Lichtenstein. Correlated metals and the LDA+U method. *Phys. Rev. B*, 67:153106–4, 2003. 54

[76] J. Pipek and P. G. Mezey. A fast intrinsic localization procedure applicable for *ab initio* and semiempirical linear combination of atomic orbital wave functions. *J. Chem. Phys.*, 90:4916, 1989. 83

[77] D. Porezag, T. Frauenheim, T. Köhler, G. Seifert, and R. Kaschner. Construction of tight-binding-like potentials on the basis of density-functional theory: Application to carbon. *Phys. Rev. B*, 51:12947, 1995. 195

[78] A. K. Rappe, C. J. Casewit, K. S. Colwell, W. A. Goddard III, and W. M. Skiff. UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *J. Am. Chem. Soc.*, 114:10024–10035, 1992. 57

[79] Anthony K. Rappé and William A. Goddard III. Charge equilibration for molecular dynamics simulation. *J. Chem. Phys.*, 95:3358–3363, 1991. 64

[80] Jan Řezáč. Empirical Self-Consistent Correction for the Description of Hydrogen Bonds in DFTB3. *J. Chem. Theory Comput.*, 13(10):4804–4817, 2017. 61, 75, 76, 77

[81] Jan Řezáč and Pavel Hobza. Advanced corrections of hydrogen bonding and dispersion for semiempirical quantum mechanical methods. *J. Chem. Theory Comput.*, 8:141–151, 2012. 61, 75, 77

[82] Kenneth Ruud and Andreas J. Thorvaldsen. Theoretical approaches to the calculation of raman optical activity spectra. *Chirality*, 21(1E):E54–E67, 2009. 137

[83] L. Medrano Sandonas, R. Gutierrez, A. Pecchia, A. Croy, and G. Cuniberti. Quantum phonon transport in nanomaterials: combining atomistic with non-equilibrium green's functions techniques. *Entropy*, 21:735, 2019. 139

[84] G. Seifert, D. Porezag, and T. Frauenheim. Calculations of molecules, clusters, and solids with a simplified LCAO-DFT-LDA scheme. *Int. J. Quant. Chem.*, 58:185, 1996. 195

[85] Grigori Sigalov, Andrew Fenley, and Alexey Onufriev. Analytical electrostatics for biomolecules: Beyond the generalized Born approximation. *J. Chem. Phys.*, 124(12):124902, 2006. 69

[86] W. Clark Still, Anna Tempczyk, Ronald C. Hawley, and Thomas Hendrickson. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.*, 112(16):6127–6129, 1990. 69

[87] Martin Stöhr, Georg S. Michelitsch, John C. Tully, Karsten Reuter, and Reinhard J. Maurer. Communication: Charge-population based dispersion interactions for molecules and materials. *J. Chem. Phys.*, 144:151101, 2016. 57, 66

[88] R Eric Stratmann, Gustavo E Scuseria, and Michael J Frisch. An efficient implementation of time-dependent density-functional theory for the calculation of excitation energies of large molecules. *J. Chem. Phys.*, 109(19):8218–8224, 1998. 90

[89] Alexandre Tkatchenko and Matthias Scheffler. Accurate Molecular Van Der Waals Interactions from Ground-State Electron Density and Free-Atom Reference Data. *Phys. Rev. Lett.*, 102:073005, 2009. 57, 66

[90] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36(5-6):232–240, June 2010. 44

[91] Stanimire Tomov, Rajib Nath, Hatem Ltaief, and Jack Dongarra. Dense linear algebra solvers for multicore with GPU accelerators. In *Proc. of the IEEE IPDPS'10*, pages 1–8, Atlanta, GA, April 19-23 2010. IEEE Computer Society. DOI: 10.1109/IPDPSW.2010.5470941. 44

[92] Y. Yang, A. Dominguez, D. Zhang, V. Lutsker, Thomas A. Niehaus, T. Frauenheim, and W. Yang. Charge transfer excitations from particle-particle random phase approximation - opportunities and challenges arising from two-electron deficient systems. *J. Chem. Phys.*, 146:124104, 2017. 91

[93] Y. Yang, H. Yu, D. York, Q. Cui, and M. Elstner. Extension of the self-consistent-charge density-functional tight-binding method: Third-order expansion of the density functional theory total energy and introduction of a modified effective coulomb interaction. *J. Phys. Chem. A*, 111:10861, 2007. 67, 75, 195

[94] V. W.-z. Yu, F. Corsetti, A. García, W. P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, Á Vázquez-Mayagoitia, C. Yang, H. Yang, and V. Blum. ELSI: A unified software interface for Kohn-Sham electronic structure solvers. *Computer Phys. Comm.*, 222:267–285, 2018. 45

[95] W. Zhang, T. S. Fisher, and N. Mingo. volume 51. Taylor and Francis, 2007. 142

[96] L. Zhechkov, Th. Heine, S. Patchkovskii, G. Seifert, and H. A. Duarte. An efficient a posteriori treatment for dispersion interaction in density-functional-based tight binding. *J. Chem. Theory Comput.*, 1:841–847, 2005. 57

[97] T. Ziegler, A. Rauk, and E. J. Baerends. Calculation of multiplet energies by the hartree-fock-slater method. *Theoretica Chimica Acta*, 43:261–271, 1977. 49

Index