User's Guide

TBtrans 4.1-b3

November 29, 2016

https://launchpad.net/siesta

Contributors to TBtrans

Contents

1 Introduction

This Reference Manual contains descriptions of all the input, output and execution features of TBTRANS*, but is not a tutorial introduction to the program.*

TBTRANS (Tight-Binding transport) is a generic computer program which calculates transport and other physical quantities using the Green function formalism. It is a stand-alone program which allows *extreme* scale tight-binding calculations.

- It uses the basic non-equilibrium Green function formalism and allows extensive customizability and analysis forms.
- TBTRANS may be given any type of local-orbital Hamiltonian and calculate transport properties of arbitrary geometries and/or number of electrodes.
- The PHTRANS variant may be compiled to obtain thermal (phonon) transport using the same Green function formalism and *all* the same functionalities as those presented in this manual.

A list of the currently implemented features are:

- Density of states (orbital resolved)
  - Green function DOS
  - Scattering DOS
- Hamiltonian interpolation at different voltages
- Selective wide-band limit of the electrode(s)
- Transmission eigenvalues
- Bulk electrode density of state and transmission (directly from the electrode Hamiltonian)
- Projected transmission of eigenstates
- Orbital resolved "bond-currents" which may subsequently be analyzed to yield actual bond-currents.

References:

- Description of the TBTRANS and TRANSIESTA code in the N terminal generic implementation [**?**].
- SISL is a data extraction utility for TBTRANS which enables easy access to the data stored in the output NetCDF-4 file [**?**].

2 Compilation

TBtrans may be compiled in the `Util/TS/TBtrans` directory.

To compile TBtrans simply go to the directory and type:

```
$ make
```

This will default to use the `arch.make` file in the `Obj` directory. To use a different directory you may do:

```
$ make OBJDIR=AnotherObjDir
```

TBtrans is tightly intertwined with the SIESTA source to reduce code duplication.

Please see the SIESTA manual for installing NetCDF easily.

2.1 The `arch.make` file

The compilation is done using a `Makefile` that is provided with the code. This `Makefile` will generate the executable for any of several architectures, with a minimum of tuning required from the user and encapsulated in a separate file called `arch.make`.

TBtrans relies on the following libraries

BLAS it is recommended to use a high-performance library (OpenBLAS or the MKL library from Intel)

- If you use your *nix distribution package manager to install BLAS you are bound to have a poor performance. Please try and use performance libraries.

To add BLAS to the `arch.make` file you need to add the required linker flags to the `LIBS` variable in the `arch.make` file.

Example variables

```
# OpenBLAS:
LIBS += -L/opt/openblas/lib -lopenblas
# or for MKL
LIBS += -L/opt/intel/.../mkl/lib/intel64 -lmkl_blas95_lp64 ...
```

LAPACK it is recommended to use a high-performance library (OpenBLAS[1] or the MKL library from Intel)

Example variables

```
# OpenBLAS (OpenBLAS will default to build in LAPACK 3.6)
LIBS += -L/opt/openblas/lib -lopenblas
# or for MKL
LIBS += -L/opt/intel/.../mkl/lib/intel64 -lmkl_lapack95_lp64 ...
```

[1] OpenBLAS enables the inclusion of the LAPACK routines. This is advised.

The above are the minimally required libraries.

Highly encouraged libraries

NetCDF Note that it should a NetCDF4 compliant compiled library[2]. This library is required for a multitude of advanced analysis methods such as orbital resolved DOS, bond-currents, $\delta\mathbf{H}$, eigenstate projections, etc.

To use this library add these variables to your `arch.make` file

```
COMP_LIBS += libncdf.a
FPPFLAGS += -DCDF -DNCDF -DNCDF_4
LIBS += -lnetcdff -lnetcdf -lhdf5_fortran -lhdf5 -lz
```

If you have compiled NetCDF4 with parallel IO you may benefit from parallel IO by adding this compilation flag:

```
FPPFLAGS += -DNCDF_PARALLEL
```

To easily install `NetCDF` please see the installation file: `Docs/install_netcdf4.bash`.

Importantly, TBtrans is compatible with hybrid parallelism using MPI and OpenMP or either of them alone.

MPI To compile using MPI add this to your `arch.make` file

```
FPPFLAGS += -DMPI
```

OpenMP To compile using OpenMP add this to your `arch.make` file

```
FFLAGS += -fopenmp
LIBS += -fopenmp
```

change the corresponding flag according to your compiler.

Running Hybrid parallel TBtrans Running TBtrans using hybrid parallelism is difficult due to the complexity of controlling the threads and processors.

To achieve good performance one *must* ensure that the threads and processors are not oversubscribe and are not overlapping. For instance if using OpenMPI $\geq$ 1.8.4 one may run TBtrans using this command:

```
mpirun -np $((PBS_NP/OMP_NUM_THREADS)) \
  -x OMP_NUM_THREADS \
  -x OMP_PROC_BIND=true \
  --map-by ppr:1:socket:pe=$OMP_NUM_THREADS
```

where `PBS_NP` is the total number of processors, `OMP_NUM_THREADS` is the number of threads per processor. The above command assumes using 1 MPI processor per socket with each socket having `OMP_NUM_THREADS` cores.

[2]Remark that a NetCDF-3 compliant library is not sufficient for TBtrans.

2.1.1 BLAS GEMM3M kernel

Several modern BLAS implementations allow the use of GEMM3M kernels for complex linear algebra. They should provide a performance enhancement for large matrices. To use these routines add this to your `arch.make` file:

```
FPPFLAGS += -DUSE_GEMM3M
```

Note that OpenMP threaded BLAS libraries are known to fail with the GEMM3M kernels.

2.1.2 Intel MKL libraries

The MKL libraries are very efficient, but may be difficult to obtain a correct linking. Here is a short tutorial for linking the MKL BLAS and LAPACK libraries correctly.

In the following assume that `MKL_ROOT` points to the root of the MKL installation directory, for instance one may install MKL into `/opt/intel/mkl`:

```
MKL_ROOT = /opt/intel/mkl
```

where `MKL_ROOT/lib/intel64` contains the libraries.

The linking depends on the used compiler:

Intel compiler The MKL libraries are parallelized using threads and you may also enable threads in TBtrans:

No threading

```
LIBS += -L$(MKL_ROOT)/lib/intel64 -lmkl_lapack95_lp64
-lmkl_blas95_lp64 -lmkl_intel_lp64 -lmkl_core -lmkl_sequential
```

OpenMP threading

```
LIBS += -openmp -L$(MKL_ROOT)/lib/intel64 -lmkl_lapack95_lp64
-lmkl_blas95_lp64 -lmkl_intel_lp64 -lmkl_core -lmkl_intel_thread
```

GNU compiler The MKL libraries are parallelized using threads and you may also enable threads in TBtrans:

No threading

```
LIBS += -L$(MKL_ROOT)/lib/intel64 -lmkl_lapack95_lp64
-lmkl_blas95_lp64 -lmkl_gf_lp64 -lmkl_core -lmkl_sequential
```

OpenMP threading

```
LIBS += -fopenmp -L$(MKL_ROOT)/lib/intel64 -lmkl_lapack95_lp64
-lmkl_blas95_lp64 -lmkl_gf_lp64 -lmkl_core -lmkl_gnu_thread
```

3 Execution of the Program

TBtrans should be called with an input file which defines what *it should do*. This may either be piped or simply added on the input line. The latter method is preferred as one may use flags for the executable.

```
$ tbtrans < RUN.fdf
$ tbtrans RUN.fdf
```

Note that if TBtrans is compiled with MPI support one may call it like

```
$ mpirun -np 4 tbtrans RUN.fdf
```

for 4 MPI-processors.

TBtrans has these optional flags:

-h print a help instruction

-L override **SystemLabel** flag

-V override **TBT.Voltage** flag. To denote the unit do as this example: **-V 0.2:eV** which sets the voltage to 0.2 eV.

-D override **TBT.Directory** flag, all output of TBtrans will be put in the corresponding folder (it will be created if non-existing)

-HS specify the **TBT.HS** variable, quickly override the used Hamiltonian

-fdf specify any given fdf flag on the command line, example **-fdf TBT.Voltage:0.2:eV**

Note that for all flags one may use ":" as a replacement for " ", although one may use quotation marks when having a space in the argument.

4 fdf-flags

Although TBtrans is a fully independent Green function transport code, it is hard-wired with the TranSIESTA fdf flags and options. If you are familiar with TranSIESTA and its input flags, then the use of TBtrans should be easy.

All fdf-flags for TBtrans are defaulted to their equivalent TranSIESTA flag. Thus if you are using TranSIESTA as a back-end you should generally not change any flags. For instance **TBT.Voltage** defaults to **TS.Voltage** if not supplied.

SystemLabel **siesta** *(string)*

The label defining this calculation. All relevant output will be prefixed with the **SystemLabel**. One may start several TBtrans calculations in the same directory if they have different labels.

TBT.Voltage 0 eV *(energy)*

Define the applied bias in the scattering region.

TBT.Directory **./** *(directory)*

Define the output directory of files from TBtrans. This allow execution of several TBtrans instances in the same folder and writing their result to different, say, sub-folders. It is particularly useful for interpolation of Hamiltonian's and for testing purposes.

TBT.Verbosity 5 *(integer)*

Specify how much information TBtrans will print-out (range 0-10).

For smaller numbers, less information will be printed, and for larger values, more information is printed.

TBT.Progress 5. *(real)*

TBtrans prints out an estimated time of completion (ETA) for the calculation. By default this is printed out every 5% of the total loops (k-point $\times$ energy loops). Setting this to 0 will print out after every energy loop.

4.1 Define electronic structure

TBT.HS <SystemLabel>.TSHS *(file)*

Define the Hamiltonian file which contains information regarding the Hamiltonian and geometry.

%block TBT.HS.Files ⟨None⟩ *(block)*

A list of files which each contain the Hamiltonian for the same geometry at different bias'. Each line has three entries, 1) the **TBT.HS** file, 2) the value of the bias applied, 3) the unit of the bias.

NOTE: if this is existing it will assume that you will perform an interpolation of the Hamiltonians to the corresponding bias (**TBT.Voltage**).

TBT.HS.Interp spline|linear *(string)*

depends on: **TBT.HS.Files**

Interpolate all files defined in **TBT.HS.Files** to the corresponding applied bias.

Generally **spline** produces the best interpolated values and its use is encouraged. The linear interpolation scheme is mainly used for comparison to the **spline**. If they are very different from each other then one may be required to perform additional self-consistent calculations at the specific bias due to large changes in the electronic structure.

Say you have calculated the SCF solution of a certain system at 5 different applied bias':

```
%block TBT.HS.Files
  ../V0/siesta.TSHS     0.  eV
  ../V-0.5/siesta.TSHS -0.5 eV
  ../V0.5/siesta.TSHS   0.5 eV
  ../V-1.0/siesta.TSHS -1.0 eV
  ../V1.0/siesta.TSHS   1.0 eV
%endblock
```

and you wish to calculate the interpolated transmissions and currents at steps of 0.1 eV, then you may use this simple loop

```
for V in `seq -1.5 0.1 1.5` ; do
   tbtrans -V $V:eV -D V$V RUN.fdf
done
```

which at each execution of TBtrans interpolates the Hamiltonian to the corresponding applied bias and store all output files in the `V$V` folder.

4.1.1 Changing the electronic structure via $\delta\mathbf{H}$

The electronic structure may be altered by changing the Hamiltonian elements via a simple additive term

$$\mathbf{H} \leftarrow \mathbf{H} + \delta\mathbf{H}, \tag{1}$$

which allows easy changes to the electronic structure or adding additional terms such as imaginary self-energies. One may also use it to add magnetic fields etc.

To use this feature at k points it is important to know that phases in TBtrans are defined using the lattice vectors (and *not* inter-atomic distances)

$$\mathbf{H}_k = \mathbf{H} \cdot e^{ik\cdot\mathbf{R}}. \tag{2}$$

TBtrans will add the phases on *all* elements of $\delta\mathbf{H}$ via Eq. (2). To counter these phases one may simply multiply $\delta\mathbf{H}$ with the negative phase $(-i)$. Note that phases are only added on super cell elements, *not* unit cell elements.

TBT.dH ⟨**None**⟩ *(file)*

Denote a file which contains the $\delta\mathbf{H}$ information. This file *must* adhere to these file format notations and is required to be supplied in a NetCDF4 format

```
netcdf file.dH {
dimensions:
        one = 1 ;
        n_s = 9 ;
        xyz = 3 ;
        no_u = 900 ;
        spin = 1 ;
variables:
        int nsc(xyz) ;
                nsc:info = "Number of supercells in each unit-cell direction" ;

group: LEVEL-1 {
  dimensions:
          nnzs = 2670 ;
  variables:
          int n_col(no_u) ;
                  n_col:info = "Number of non-zero elements per row" ;
          int list_col(nnzs) ;
                  list_col:info = "Supercell column indices in the sparse format" ;
          int isc_off(n_s, xyz) ;
                  isc_off:info = "Index of supercell coordinates" ;
          double RedH(spin, nnzs) ;
                  RedH:info = "Real part of dH" ;
                  RedH:unit = "Ry" ;
          double ImdH(spin, nnzs) ;
                  ImdH:info = "Imaginary part of dH" ;
                  ImdH:unit = "Ry" ;
  } // group LEVEL-1

group: LEVEL-2 {
  dimensions:
```

```
            nkpt = UNLIMITED ;
            nnzs = 2670 ;
    variables:
            double kpt(nkpt, xyz) ;
                    kpt:info = "k-points for dH values" ;
                    kpt:unit = "b**-1" ;
            ... n_col list_col isc_off ...
            double dH(nkpt, spin, nnzs) ;
                    dH:info = "dH" ;
                    dH:unit = "Ry" ;
    } // group LEVEL-2

  group: LEVEL-3 {
    dimensions:
            ne = UNLIMITED ;
            nnzs = 2670 ;
    variables:
            double E(ne) ;
                    E:info = "Energy points for dH values" ;
                    E:unit = "Ry" ;
            ... n_col list_col isc_off ...
            double dH(ne, spin, nnzs) ;
                    dH:info = "dH" ;
                    dH:unit = "Ry" ;
    } // group LEVEL-3

  group: LEVEL-4 {
    dimensions:
            nkpt = UNLIMITED ;
            ne = UNLIMITED ;
            nnzs = 2670 ;
    variables:
            double kpt(nkpt, xyz) ;
                    kpt:info = "k-points for dH values" ;
                    kpt:unit = "b**-1" ;
            double E(ne) ;
                    E:info = "Energy points for dH values" ;
                    E:unit = "Ry" ;
            ... n_col list_col isc_off ...
            double dH(nkpt, ne, spin, nnzs) ;
                    dH:info = "dH" ;
                    dH:unit = "Ry" ;
    } // group LEVEL-4
  }
```

This example file shows how the file should be formatted. Note that one may either define the Hamiltonian as `dH` or as `RedH` and `ImdH`. The former is defining $\delta\mathbf{H}$ as a real quantity while the latter makes it an imaginary $\delta\mathbf{H}$.

The levels are defined because they have precedence from each other, if the energy point and k point is found in LEVEL-4 it will use this, if not, it will check for the energy point in LEVEL-3, and so on.

The remaining options are only applicable if **TBT.dH** has been set.

TBT.dH.Parallel **true** *(logical)*

Whether the $\delta\mathbf{H}$ file should be read in parallel. If your architecture supports parallel IO it is beneficial to do so. TBTRANS performs a basic check whether parallel IO may be possible, if it cannot assert this it will be turned off.

TBT.dH.Current.Orb **true** *(logical)*

depends on: **TBT.Current.Orb**

Whether the orbital currents will also contain the $\delta\mathbf{H}$ contributions.

If **false** the bond-currents will be calculated with-out the additional term arising from the $\delta\mathbf{H}$. This may be useful in cases where the additional term arises due to self-energy corrections.

NOTE: this functionality will only work correctly with a sorted sparse $\delta\mathbf{H}$ entry. For every row the column indices must be sorted.

4.2 Determine calculated physical quantities

TBTRANS can calculate a large variety of physical quantities. By default it will only calculate the transmission between the electrodes. Calculating as few quantities as possible will increase throughput, while requesting many quantities will result in much longer run-times.

You are heavily encouraged to compile TBTRANS with NetCDF4 support, see Sec. 2.1, as quantities will be orbital resolved.

If TBTRANS has been compiled with NetCDF4 support, one may extract the projected DOS from the `SystemLabel.TBT.nc` using SISL (or manual scripting). The calculated DOS can *only* be extracted from the atoms in the device region (atoms in block **TBT.Atoms.Device**). Hence the **TBT.Atoms.Device** block is *extremely* important when conducting detailed DOS analysis. For instance if the input file has this:

```
%block TBT.Atoms.Device
  atom [20 -- 40]
%endblock
```

one may extract the PDOS on a subset of atoms using this SISL command

```
sdata siesta.TBT.nc --atom 20-30 --dos --ados Left --out dos_20-30.dat
sdata siesta.TBT.nc --atom 20-30[1-3] --dos --ados Left --out dos_20-30_1-3.dat
```

where the former is the total PDOS on atoms 20 through 30, and the latter is the PDOS on orbitals 1, 2 and 3 on atoms 20 through 30. It thus is extremely easy to extract different PDOS once the calculation has completed.

TBT.T.Bulk **false** *(logical)*

Calculate the bulk (pristine) electrode transmission if **true**.

This generates `SystemLabel.BTRANS_<>` and `SystemLabel.AVBTRANS_<>`.

NOTE: implicitly enables **TBT.DOS.Elecs** if **true**.

TBT.DOS.Elecs **false** *(logical)*

Calculate the bulk (pristine) electrode DOS if **true**.

This generates `SystemLabel.BDOS_<>` and `SystemLabel.AVBDOS_<>`.

NOTE: implicitly enables **TBT.T.Bulk** if **true**.

TBT.DOS.Gf **false** *(logical)*

depends on: **TBT.Atoms.Device**

Calculate the DOS from the Green function on the atoms in the device region.

NOTE: this flag should only be used if there are bound states in the scattering region (or if one wish to uncover whether there are bound states). Due to internal algorithms the DOS from the Green function is computationally more demanding than using **TBT.DOS.A** an **TBT.DOS.A.All**.

This generates `SystemLabel.DOS` and `SystemLabel.AVDOS`.

See **TBT.Atoms.Device.Connect**.

TBT.DOS.A **false** *(logical)*

depends on: **TBT.Atoms.Device**

Calculate the DOS from the spectral function. This will not calculate the DOS from the last electrode (last in the list **TBT.Elecs**), see **TBT.DOS.A.All**.

This generates `SystemLabel.ADOS_<>` and `SystemLabel.AVADOS_<>`.

See **TBT.Atoms.Device.Connect**.

TBT.DOS.A.All **false** *(logical)*

depends on: **TBT.Atoms.Device**

Calculate the DOS from the spectral function and do so with *all* electrodes.

This additionally generates `SystemLabel.ADOS_<>` and `SystemLabel.AVADOS_<>` for the last electrode in **TBT.Elecs**.

NOTE: if **true**, this implicitly sets **TBT.DOS.A** to **true**.

Setting the flags **TBT.DOS.Gf** and **TBT.DOS.A.All** to **true** enables the estimation of bound states in the scattering region via this simple expression

$$\rho_{\mathrm{bound-states}} = \rho_{\mathbf{G}} - \sum_i \rho_{\mathbf{A}_i}, \tag{3}$$

where the sum is over all electrodes, $\mathbf{G}$ and $\mathbf{A}$ are the Green and spectral function, respectively. Note that typically $\rho_{\mathrm{bound-states}} = 0$.

TBT.T.Eig **0** *(integer)*

Specify how many of the transmission eigenvalues will be calculated.

This generates `SystemLabel.TEIG_<1>_<2>` and `SystemLabel.AVTEIG_<1>_<2>`, however, only for the non-equivalent electrode combinations (TBtrans intrinsically assumes time-reversal symmetry).

NOTE: if you specify a number of eigenvalues above the available number of eigenvalues, TBtrans will automatically truncate it to a reasonable number.

TBT.T.All **false** *(logical)*

By default TBtrans only calculates transmissions in *one* direction because time-reversal symmetry makes $T_{ij} = T_{ji}$. If one wishes to assert this, or if time-reversal symmetry does not apply for your system, one may set this to **true** to explicitly calculate all transmissions.

This additionally generates `SystemLabel.TRANS_<1>_<2>` and `SystemLabel.AVTRANS_<1>_<2>` for all electrode combinations (and the equivalent eigenvalue files if **TBT.T.Eig** is **true**.

TBT.T.Out **false** *(logical)*

The total transmission out of any electrode[3] may easily be calculated using only the scattering matrix of the origin electrode and the scattering region Green function. This enables the calculation of these equations

$$i\,\mathrm{Tr}[(\mathbf{G}-\mathbf{G}^\dagger)\mathbf{\Gamma}_j], \tag{4}$$

$$\mathrm{Tr}[\mathbf{G}\mathbf{\Gamma}_j\mathbf{G}^\dagger\mathbf{\Gamma}_j]. \tag{5}$$

The total transmission out of electrode j may then be calculated as

$$T_j = i\,\mathrm{Tr}[(\mathbf{G}-\mathbf{G}^\dagger)\mathbf{\Gamma}_j] - \mathrm{Tr}[\mathbf{G}\mathbf{\Gamma}_j\mathbf{G}^\dagger\mathbf{\Gamma}_j]. \tag{6}$$

This generates two sets of files: `SystemLabel.CORR_<>` and `SystemLabel.TRANS_<1>_<1>` which corresponds to equations Eqs. (4) and (5), respectively. To calculate T_j subtract the two files according to Eq. (6).

TBT.Current.Orb **false** *(logical)*

depends on: **TBT.Atoms.Device**, **TBT.DOS.A**

Whether the orbital currents will be calculated and stored. These will be stored in a sparse matrix format corresponding to the SIESTA sparse format with only the device atoms in the sparse pattern.

Orbital currents are implemented as:

$$J_{\alpha\beta} = i[\mathbf{H}_{\beta\alpha}\mathbf{A}_{\alpha\beta} - \mathbf{H}_{\alpha\beta}\mathbf{A}_{\beta\alpha}], \tag{7}$$

where we have left out the pre-factor $(e/\hbar)$ intentionally. SISL may be used to analyze the orbital currents and enables easy transformation of orbital currents to bond currents and activity currents.

NOTE: this requires TBTRANS to be compiled with NetCDF-4 support, see Sec. 2.1.

TBT.Spin ⟨**all**⟩ *(integer)*

If the Hamiltonian is a polarized calculation one my define the index of the spin to be calculated.

This allows one to simultaneously calculate the spin-up and spin-down transmissions, for instance

```
$ tbtrans -fdf TBT.Spin:1 -D UP RUN.fdf &
$ tbtrans -fdf TBT.Spin:2 -D DOWN RUN.fdf &
```

which will create two folders `UP` and `DOWN` and output the relevant physical quantities in the respective folders.

TBT.Symmetry.TimeReversal **true** *(logical)*

Whether the Hamiltonian and the calculation should use time-reversal symmetry. Currently this only affects **k**-point sampling calculations by not removing any symmetry **k**-points.

If one has **k**-point sampling and wishes to use **TBT.Current.Orb** this should be **false**.

[3] In $N > 2$-electrode calculations one *cannot* use this quantity to calculate the total current out of an electrode.

4.2.1 Device region

The scattering region (and thus device region) is formally consisting of all atoms besides the electrodes. However, when calculating the transmission this choice is very inefficient. Thus to heavily increase throughput one may define a smaller device region consisting of a subset of atoms in the scattering region.

The choice of atoms *must* separate each electrode from each other. TBtrans will stop if this is not enforced.

Remark that the physical quantities such as DOS, spectral DOS, orbital currents may only be calculated in the selected device region.

TBT.Atoms.Device ⟨**all but electrodes**⟩ *(block/list)*

This flag may either be a block, or a list.

A block with each line denoting the atoms that consists of the device region.

```
%block TBT.Atoms.Device
   atom [ 10 -- 20 ]
   atom [ 30 -- 40 ]
% endblock
# Or equivalently as a list
TBT.Atoms.Device [10 -- 20, 30 -- 40]
```

will limit the device region to atoms [10–20, 30–40].

TBT.Atoms.Device.Connect **false** *(logical)*

Whether the device region is extended such that the DOS is calculated correctly on the device defined atoms.

NOTE: this parameter should be set to **true** in case accurate DOS calculations are required on the specified device atoms.

TBT.Atoms.Buffer ⟨**None**⟩ *(block/list)*

A block with each line denoting the atoms that are disregarded in the Green function calculation. For self-consistent calculations it may be required to introduce buffer atoms which are removed from the SCF cycle. In such cases these atoms should also be removed from the transport calculation.

```
%block TBT.Atoms.Buffer
   atom [ 1 -- 5 ]
%endblock
# Or equivalently as a list
TBT.Atoms.Buffer [1 -- 5]
```

will remove atoms [1–5] from the calculation.

4.2.2 Brillouin zone

TBtrans allows calculating physical quantities via averaging in the Brillouin zone.

TBT.k ⟨**kgrid_Monkhorst_Pack**⟩ *(list/block)*

Specify how to perform Brillouin zone integrations.

This may be given as a list like this:

```
TBT.k [A B C]
```

where each integer corresponds to the diagonal elements of the Monkhorst-Pack grid. I.e.

```
TBT.k [10 10 1]
%block TBT.k
  10  0 0 0.
   0 10 0 0.
   0  0 1 0.
%endblock
```

are equivalent.

If you supply this flag as a block the following options are available:

path Define a Brillouin zone path[4] where the k-points are equi-spaced. It may be best described using this example:

```
path 10
  from 0.  0.  0.
  to   0.5 0.5 0.
path 20
  from 0.25 0.25 0.
  to   0.0  0.5  0.
```

This will create k-points starting from the Γ-point and move to the Brillouin zone boundary at [1/2, 1/2, 0] with spacing to have 10 points.

There is no requirement that the **path**s are connected and one may specify as many paths as wanted.

even-path It is generally advised to add this flag in the blog (somewhere) if one wants equi-distance k-spacings in the Brillouin zone. This flag sums up the total number of k-points on the total path and then calculates the exact number of required points required on each path to have the same δk in each path.

NOTE: if any one **path** is found in the block the options (explained below) are ignored.

diagonal|diag Specify the number of k points in each unit-cell direction

diagonal 3 3 1 will use 3 k points along the first and second lattice vectors and only one along the third lattice vector.

displacement|displ Specify the displacement of the Brillouin zone k points along each lattice vector. This input is similar to **diagonal** but requires real input.

displacement 0.5 0.25 0. will displace the first and second k origin to [1/2, 1/4, 0].

size This reduces the sampled Brillouin zone to only the fractional size of each lattice vector direction.

This may be used to only sample k-points in a reduced Brillouin zone which for instance is useful if one wishes to sample the Dirac point in graphene in an energy range of $-0.5\,\mathrm{eV}$ – $0.5\,\mathrm{eV}$.

size 0.5 1. 1. will reduce the sampled k points along the first reciprocal lattice to be in the range $]-1/4, 1/4]$, while the other directions are still sampled $]-1/2, 1/2]$.

[4] Much like **BandLines** in SIESTA.

NOTE: expert use only.

list Explicitly specify the sampled k-points and (optionally) the associated weights.

```
list 2
  0.  0.  0.  0.5
  0.5 0.5 0.
```

where the integer on the **list** line specifies the number of lines that contains k points. Each line *must* be created with 3 reals which define the k point in units of the reciprocal lattice vectors (]−1/2–1/2]).

An optional 4th value denote the associated weight which is defaulted to $1/N$ where N is the total number of k points.

NOTE: if this is found it will neglect the other input options (except **path**).

method Define how the k-points should be created in the Brillouin zone.

Currently these options are available (**Monkhorst-Pack** being the default)

Monkhorst-Pack|MP Use the regular Monkhorst-Pack sampling (equi-spaced) with simple linear weights.

Gauss-Legendre Use the Gauss-Legendre quadrature and weights for constructing the k points and weights. These k points are not equi-spaced and puts more weight to the Γ point.

Simpson-mix Use the Newton-Cotes method (Simpson, degree 3) which uses equi-spaced points but non-uniform weights.

Boole-mix Use the Newton-Cotes method (Boole, degree 5) which uses equi-spaced points but non-uniform weights.

⟨**siesta-method**⟩ One may also use the typical **kgrid_Monkhorst_Pack** method of input as done in SIESTA. This is a 3×3 block such as:

```
10  0  0 0.
 0 15  0 0.
 0  0  1 0.
```

which uses 10, 15 and 1 k-points along the 1st, 2nd and 3rd reciprocal lattice vectors. And with 0 displacement.

NOTE: it is recommended to use the **diagonal** option unless off-diagonal k points are needed.

4.2.3 Energy grid

The Green function is calculated at explicit energies and does not rely on diagonalization routines to retrieve the eigenspectrum. This is due to the smearing of states from the coupling with the semi-infinite electrodes.

It is thus important to define an energy grid for analysis of the DOS and transmission.

TBT.Contours.Eta 0 eV *(energy)*

The imaginary (η) part of the Green function in the device region. Note that the electrodes imaginary part may be controlled via **TBT.Elecs.Eta** and **TBT.Elec.<>.tbt.Eta**.

This value controls the smearing of the DOS on the energy axis. Generally will the electrode η

value contribute to a smearing in the device region, while in certain situations an imaginary η is required in the device region.

%block TBT.Contours *see note further down* *(block)*

Each line in this block corresponds to a specific contour. Enabling several lines of input allows to create regions of the energy grid which has a high density and ranges of energies with lower density. Also it allows to bypass energy ranges where the DOS is zero in for instance a semiconductor.

See **TBT.Contour.<>** for details on specifying the energy contour.

%block TBT.Contour.<> ⟨**None**⟩ *(block)*

Specify a contour named **<>** with options within the block.

The names **<>** are taken from the **TBT.Contours** block.

The format of this block is made up of at least 3 lines, in the following order of appearance.

from *a* to *b* Define the integration range on the energy axis. Thus a and b are energies.

points/delta Define the number of integration points/energy separation. If specifying the number of points an integer should be supplied.

If specifying the separation between consecutive points an energy should be supplied (e.g. **0.01 eV**).

method Specify the numerical method used to conduct the integration. Here a number of different numerical integration schemes are accessible

mid|mid-rule Use the mid-rule for integration.

simpson|simpson-mix Use the composite Simpson 3/8 rule (three point Newton-Cotes).

boole|boole-mix Use the composite Booles rule (five point Newton-Cotes).

G-legendre Gauss-Legendre quadrature.

tanh-sinh Tanh-Sinh quadrature.

NOTE: has **opt precision <>**.

opt Specify additional options for the **method**. Only a selected subset of the methods have additional options.

By default the TBtrans energy grid is defined as

```
%block TBT.Contours
  line
%endblock TBT.Contours
%block TBT.Contour.line
   from -2. eV to 2. eV
    delta 0.01 eV
      method mid-rule
%endblock TBT.Contour.line
```

4.3 Chemical potentials

For N electrodes there will also be N_μ chemical potentials. They are defined via blocks similar to **TBT.Elecs**. If no bias is applied TBtrans will default to a single chemical potential with the chemical potential in equilibrium. In this case you need not specify any chemical potentials.

By default TBtrans creates a single chemical potential with the chemical potential equal to the device Fermi-level. Hence, performing non-bias calculations does not require one to specify these blocks.

%block TBT.ChemPots ⟨**None**⟩ *(block)*

Each line denotes a new chemical potential which may is further defined in the **TBT.ChemPot.<>** block.

%block TBT.ChemPot.<> ⟨**None**⟩ *(block)*

Each line defines a setting for the chemical potential named **<>**.

chemical-shift|mu Define the chemical shift (an energy) for this chemical potential. One may specify the shift in terms of the applied bias using **V/<integer>** instead of explicitly typing the energy.

ElectronicTemperature|Temp|kT Specify the electronic temperature (as an energy or in Kelvin). This defaults to **TS.ElectronicTemperature**.

One may specify this in units of **TS.ElectronicTemperature** by using the unit **kT**.

It is important to realize that the parameterization of the voltage into the chemical potentials enables one to have a *single* input file which is never required to be changed, even when changing the applied bias.

These options complicate the input sequence for regular 2 electrode which is unfortunate.

4.4 Electrode configuration

The electrodes are defining the semi-infinite region that is coupled to the scattering region.

TBtrans is a fully N electrode calculator. Thus the input for such setups is rather complicated.

TBtrans defaults to read the TranSIESTA electrodes and as such one may replace **TBT** by **TS** and TBtrans will still work. However, the **TBT** has precedence.

If there is only 1 chemical potential all electrodes will default to use this chemical potential, thus for non-bias calculations there is no need to specify the chemical potential (**TBT.Elec.<>.chemical-potential**).

%block TBT.Elecs ⟨**None**⟩ *(block)*

Each line denote an electrode which may be queried in **TBT.Elec.<>** for its setup.

%block TBT.Elec.<> ⟨**None**⟩ *(block)*

Each line represents a setting for electrode **<>**. There are a few lines that *must* be present, **HS**, **semi-inf-dir**, **electrode-pos**, **chem-pot** (only if **TBT.Voltage** is not 0).

HS The electronic structure information from the initial electrode calculation. This file retains

the geometrical information as well as the Hamiltonian, overlap matrix and the Fermi-level of the electrode. This is a file-path and the electrode `SystemLabel.TSHS` need not be located in the simulation folder.

TBTRANS also reads NetCDF4 files which contain the electronic structure. This may be created using SISL.

semi-inf-direction|semi-inf-dir|semi-inf The semi-infinite direction of the electrode with respect to the electrode unit-cell.

NOTE: this has nothing to do with the scattering region unit cell, TRANSIESTA will figure out the alignment of the electrode unit-cell and the scattering region unit-cell.

chemical-potential|chem-pot|mu The chemical potential that is associated with this electrode. This is a string that should be present in the **TBT.ChemPots** block in case there is a bias applied in the calculation.

electrode-position|elec-pos The index of the electrode in the scattering region. This may be given by either **elec-pos <idx>**, which refers to the first atomic index of the electrode residing at index **<idx>**. Else the electrode position may be given via **elec-pos end <idx>** where the last index of the electrode will be located at **<idx>**.

used-atoms Number of atoms from the electrode calculation that is used in the scattering region as electrode. This may be useful when the periodicity of the electrodes forces extensive electrodes in the semi-infinite direction.

NOTE: do not set this if you use all atoms in the electrode.

Bulk Logical controlling whether the Hamiltonian of the electrode region in the scattering region is enforced *bulk* or whether the Hamiltonian is taken from the scattering region elements.

tbt.Gf/Gf String with filename of the surface Green function data. This may be used to place a common surface Green function file in a top directory which may then be used in all calculations using the same electrode and the same contour. If many calculations are performed this will heavily increase performance at the cost of disk-space.

tbt.Eta/Eta Control the imaginary part of the surface Green function for this electrode. See **TBT.Elecs.Eta**.

tbt.Accuracy/Accuracy Control the convergence accuracy required for the self-energy calculation when using the Lopez-Sanchez, Lopez-Sanchez iterative scheme. See **TBT.Elecs.Accuracy**.

NOTE: advanced use *only*.

Bloch 3 integers are present on this line which each denote the number of times bigger the scattering region electrode is compared to the electrode, in each lattice direction. Remark that these expansion coefficients are with regard to the electrode unit-cell. This is denoted "Bloch" because it is an expansion based on Bloch waves.

Bloch-A/a1|B/a2|C/a3 Specific Bloch expansions in each of the electrode unit-cell direction. See **Bloch** for details.

pre-expand String denoting how the expansion of the surface Green function file will be performed. This only affects the Green function file if **Bloch** is larger than 1. By default the Green function file will contain the fully expanded surface Green function, Hamiltonian and

overlap matrices (**all**). One may reduce the file size by setting this to **Green** which only expands the surface Green function. Finally **none** may be passed to reduce the file size to the bare minimum. For performance reasons **all** is preferred.

tbt.out-of-core/out-of-core If **true** the GF files are created which contain the surface Green function. Setting this to **true** may be advantageous when performing many calculations using the same k and energy grid using the same electrode. In those case this will heavily increase throughput. If **false** (default) the surface Green function will be calculated when needed.

NOTE: simultaneous calculations may read the same GF file.

tbt.Gf-Reuse *depends on:* **TBT.Elec.<>.tbt.out-of-core**

Logical deciding whether the surface Green function file should be re-used or deleted. If this is **false** the surface Green function file is deleted and re-created upon start.

tbt.check-kgrid For N electrode calculations the **k** mesh will sometimes not be equivalent for the electrodes and the device region calculations. However, TBtrans requires that the device and electrode **k** samplings are commensurate. This flag controls whether this check is enforced.

NOTE: only use if fully aware of the implications (for tight-binding calculations this may safely be set to **false**).

There are several flags which are globally controlling the variables for the electrodes (with **TBT.Elec.<>** taking precedence).

TBT.Elecs.Bulk **true** *(logical)*

This globally controls how the Hamiltonian is treated in all electrodes. See **TBT.Elec.<>.Bulk**.

TBT.Elecs.Eta $10^{-4}\,\mathrm{eV}$ *(energy)*

Globally control the imaginary part used for the surface Green function calculation. This η value is *not* used in the device region. See **TBT.Elec.<>.tbt.Eta**.

TBT.Elecs.Accuracy $10^{-13}\,\mathrm{eV}$ *(energy)*

Globally control the accuracy required for convergence of the self-energy. See **TBT.Elec.<>.tbt.Accuracy**.

TBT.Elecs.Neglect.Principal **false** *(logical)*

If this is **false** TranSIESTA dies if there are connections beyond the principal cell.

NOTE: set this to **true** with care, non-physical results may arise. Use at your own risk!

TBT.Elecs.Out-of-core **false** *(logical)*

This enables reusing the self-energies by storing them on-disk (**true**). The surface Green function files may be large files but heavily increases throughput if one performs several transport calculations using the same electrodes.

You are encouraged to set this to **true** to reduce computations. See **TBT.Elec.<>.tbt.out-of-core**.

Currently this option is not compatible with **TBT.T.Bulk** and **TBT.DOS.Elecs**, and the bulk transmission and bulk DOS will not be calculated if this option is **true**.

TBT.Elecs.Gf.Reuse **true** *(logical)*

depends on: **TBT.Elecs.Out-of-core**, **TBT.Elec.<>.tbt.out-of-core**

Globally control whether the surface Green function files should be re-used (**true**) or re-created (**false**). See **TBT.Elec.<>.tbt.Gf-Reuse**.

TBT.Elecs.Coord.EPS 10^{-4} Bohr *(length)*

When using Bloch expansion of the self-energies one may experience difficulties in obtaining perfectly aligned electrode coordinates.

This parameter controls how strict the criteria for equivalent atomic coordinates is. If TRANSIESTA crashes due to mismatch between the electrode atomic coordinates and the scattering region calculation, one may increase this criteria. This should only be done if one is sure that the atomic coordinates are almost similar and that the difference in electronic structures of the two may be negligible.

4.4.1 Principal layer interactions

It is *extremely* important that the electrodes only interact with one neighboring supercell due to the self-energy calculation. TBTRANS will print out a block as this

```
<> principal cell is perfect!
```

if the electrode is correctly setup and it only interacts with its neighboring supercell. In case the electrode is erroneously setup, something similar to the following will be shown in the output file.

```
<> principal cell is extending out with 96 elements:
   Atom 1 connects with atom 3
   Orbital 8 connects with orbital 26
   Hamiltonian value: |H(8,6587)|@R=-2 =  0.651E-13 eV
   Overlap          :  S(8,6587)|@R=-2 =   0.00
```

It is imperative that you have a *perfect* electrode as otherwise nonphysical results will occur.

4.5 Calculation settings

The calculation time is currently governed by two things:

1. the size of the device region,
2. and by the partitioning of the block-tri-diagonal matrix.

The first may be controlled via **TBT.Atoms.Device**. If one is only interested in transmission coefficients this flag is encouraged to select the minimum number of atoms that will successfully run the calculation. Please see the flag entry for further details.

Secondly there is, currently, no way to determine the most optimal block-partitioning of a banded matrix and TBTRANS allows several algorithms to determine an optimal partitioning scheme. The following flag controls the partitioning for the device region.

TBT.BTD.Pivot.Device **atom-⟨first electrode⟩** *(string)*

Decide on the partitioning for the BTD matrix. One may denote either **atom+** or **orb+** as a prefix which does the analysis on the atomic sparsity pattern or the full orbital sparsity pattern, respectively. If neither are used it will default to **atom+**.

<elec-name> The partitioning will be a connectivity graph starting from the electrode denoted by the name. This name *must* be found in the **TBT.Elecs** block.

NOTE: One may append an optional setting **front** or **fan** which makes the connectivity graph to consider the geometric front of the atoms. For extreme scale simulations or tight-binding calculations with constrictions this may improve the BTD matrix substantially because it splits the unit-cell into segments of equal width.

rev-CM Use the reverse Cuthill-McKee for pivoting the matrix elements to reduce bandwidth. One may omit **rev-** to use the standard Cuthill-McKee algorithm.

GPS Use the Gibbs-Poole-Stockmeyer algorithm for reducing the bandwidth.

GGPS Use the generalized Gibbs-Poole-Stockmeyer algorithm for reducing the bandwidth.

PCG Use the perphiral connectivity graph algorithm for reducing the bandwidth.

Examples are

```
TBT.BTD.Pivot.Device atom+GGPS
TBT.BTD.Pivot.Device GGPS
TBT.BTD.Pivot.Device orb+GGPS
TBT.BTD.Pivot.Device orb+PCG
```

where the first two are equivalent. The 3rd and 4th are more heavily on analysis and will typically not improve the bandwidth reduction.

TBT.BTD.Pivot.Elec.<> **atom-⟨<>⟩** *(string)*

depends on: **TBT.Atoms.Device**

If **TBT.Atoms.Device** has been set to a reduced region the electrode self-energies must be *down-folded* through the atoms not part of the device-region. In this case these down-fold regions can also be considered a BTD matrix which may be optimized separately from the device region BTD matrix.

This option have all available options as described in **TBT.BTD.Pivot.Device** but one will generally find the best pivoting scheme by using the default (**atom-<>**) which is the atomic connectivity graph from the electrode it-self.

It may be advantageous to use **atom-<>-front** for very large tight-binding calculations where the device region is chosen far from this electrode and normal to the electrode-plane.

4.6 Input/Output

TBtrans IO is mainly relying on the NetCDF4 library and full capability is only achieved if compiled with this library.

Several fdf-flags control how TBtrans performs IO.

TBT.CDF.Precision **single|float|double** *(string)*

Specify the precision used for storing the quantities in the NetCDF4.

single takes half the disk-space as **double** and will generally retain a sufficient precision of the quantities.

single and **float** are equivalent.

NOTE: all calculations are performed using **double** so this is *only* a storage precision.

TBT.CDF.DOS.Precision <TBT.CDF.Precision> *(string)*

Specify the precision used for storing DOS in NetCDF4.

See **TBT.CDF.Precision**.

TBT.CDF.T.Precision <TBT.CDF.Precision> *(string)*

Specify the precision used for storing transmission function in NetCDF4.

See **TBT.CDF.Precision**.

TBT.CDF.T.Eig.Precision <TBT.CDF.Precision> *(string)*

Specify the precision used for storing transmission eigenvalues in NetCDF4.

See **TBT.CDF.Precision**.

TBT.CDF.Current.Precision <TBT.CDF.Precision> *(string)*

Specify the precision used for storing orbital current in NetCDF4.

See **TBT.CDF.Precision**.

NOTE: This is heavily advised to be in single precision as this may easily use large amounts of disk-space if in double precision.

TBT.CDF.Compress 0 *(integer)*

Specify whether the NetCDF4 files stored will be compressed. This may heavily reduce disk-utilization at the cost of some performance.

This number must be between 0 (no compression) and 9 (maximum compression). A higher compression is more time consuming and a good compromise between speed and compression is 3.

NOTE: one may subsequently to a TBtrans compilation compress a NetCDF4 file using:

```
nccopy -d 3 siesta.TBT.nc newsiesta.TBT.nc
```

TBT.CDF.MPI false *(logical)*

Whether the IO is performed in parallel. If using a large amount of MPI processors this may increase performance.

NOTE: this automatically sets the compression to 0 (one cannot compress and perform parallel IO)

4.6.1 Self-energy

TBtrans enables the storage of the self-energies from the electrodes in selected regions. I.e. in a two electrode setup the self-energies may be "down-folded" to a region of interest (say molecule etc.) and then saved.

This feature enables one to easily use self-energies in Python for subsequent analysis etc. It is only available if compiled against NetCDF4.

TBT.CDF.SelfEnergy.Save **false** *(logical)*

Store the self-energies of the electrodes. The self-energies are first down-folded into the device region (see **TBT.Atoms.Device**).

TBT.CDF.SelfEnergy.Precision **<TBT.CDF.Precision>** *(string)*

Specify the precision used for storing the self-energies in NetCDF4.

See **TBT.CDF.Precision**.

TBT.CDF.SelfEnergy.Only **false** *(logical)*

If **true** this will *only* calculate and store the down-folded self-energies. No physical quantities will be calculated and TBtrans will quit.

TBT.CDF.SelfEnergy.Compress **<TBT.CDF.Compress>** *(integer)*

Specify the compression of the self-energies in NetCDF4.

See **TBT.CDF.Compress**.

4.6.2 Projected transmissions

The transmission through a scattering region is determined by the electrodes band-structure and the energy levels for the scattering part. In for instance molecular electronics it is often useful to determine which molecular orbitals are responsible for the transmission as well as knowing their hybridization with the substrate (electrodes).

TBtrans implements an advanced projection method which splits the transmission and DOS into eigenstate projectors.

In the following we concentrate on a 2 terminal device while it may be used for N electrode calculations. One important aspect of projection is that the self-energies that are to be projected *must* be fully located on the projection region. TBtrans will die if this is not enforced. A projection can *only* be performed if the down-folding of the self-energies for the projected electrode is fully encapsulated in the device region (**TBT.Atoms.Device**). I.e. one should reduce the device region such that any couplings from the electrodes only couple into the projection region. Generally for the most simple projections the device region should be equivalent to the projection region in case there is only one projection region.

These projections should not be confused with local DOS which may be obtained if compiled with the NetCDF4 library and via the use of sisl, see Sec. 4.2.

NOTE: if the **TBT.Projs** block is defined, then the **TBT.Projs.T** block is required in the input unless **TBT.Projs.Init** is **true**.

%block TBT.Projs ⟨**None**⟩ *(block)*

List of molecular projections used:

```
%block TBT.Projs
  M-L
  M-R
%endblock
```

This tells TBtrans that two projections will exist. Each projection setup will be read in **TBT.Proj.<>**.

There is no limit to the number of projection molecules.

%block TBT.Proj.<> ⟨**None**⟩ *(block)*

Block that designates a molecular projection by the names specified in the **TBT.Projs** block.

This block determines how each projection is interpreted, it consists of several options defined below:

atom There may be several **atom** lines. The full set of atomic indices will be used as a sub-space for the Hamiltonian. The atoms may be defined via these variants

atom *A* [***B*** [***C*** [**...**]]] A sequence of atomic indices which are used for the projection.

atom from *A* to *B* [**step *s***] Here atoms A up to and including B are used. If **step <s>** is given, the range A:B will be taken in steps of s.

```
atom from 3 to 10 step 2
```

will add atoms 3, 5, 7 and 9.

atom from *A* plus/minus *B* [**step *s***] Atoms A up to and including $A + B - 1$ are added to the projection. If **step <s>** is given, the range A:$A + B - 1$ will be taken in steps of s.

atom [**<A>, *B* -- *C*** [**step *s***], ***D***] Equivalent to **from ... to** specification, however in a shorter variant. Note that the list may contain arbitrary number of ranges and/or individual indices.

```
atom [2, 3 -- 10 step 2, 6]
```

will add atoms 2, 3, 5, 7, 9 and 6.

Gamma Logical variable which determines whether the projectors are the Γ-point projectors, or the k resolved ones. For Γ-only calculations this has no effect. If the eigenstates are non-dispersive in the Brillouin zone there should be no difference between **true** or **false**.

NOTE: it is *very* important to know the dispersion and possible band-crossings of the eigenstates if this option is **false**. For band-crossings one must manually perform the projections for the k-points in a stringent manner as the order of eigenstates are not retained.

proj <P-name> Allows to define a projection based on the eigenstates for the current molecule.

The **<P-name>** designates the name associated with this projection.

It is parsed like this, in the following 0 is the Fermi level (HOMO = −1, LUMO = 1):

level from <E1> to <E2> Energy eigenstates **E1** and **E2** will be part of the molecular orbitals that constitute this projection

level from <E> plus <N> Energy eigenstates between **E** and **E** + N − 1 will be part of the molecular orbitals that constitute this projection

level from <E> minus <N> Energy eigenstates between **E** and **E** − N + 1 will be part of the molecular orbitals that constitute this projection

level <E1> <E2> ... <En> All eigenstates specified will be part of the molecular orbitals that constitute this projection

level [**<list>**] A comma-separated list specification.

end All gathered eigenstates so far will constitute the projection named **<P-name>**

Note that level 0 refers to the Fermi level, it will be silently removed as it is not an eigenstate, so you do not need to think about it.

You can specify named projection blocks as many times as you want.

To conclude the full projection block here is an example describing three different projections for the left molecule in

```
%block TBT.Proj.M-L
  # We have 2 atoms on this molecule
  atom from 5 plus 2
  # We only do a Gamma projection
  Gamma .true.
  # We will utilise three different projections on
  # this molecule
  proj HOMO
   level -1
  end
  proj LUMO
   level 1
  end
  proj H-plus-L
   level from -1 to 1
  end
%endblock
```

Similarly for the right molecule we do

```
%block TBT.Proj.M-R
  # We have 2 atoms on this molecule
  atom from 8 plus 2
  # We only do a Gamma projection
  Gamma .true.
  # We will utilise three different projections on
  # this molecule
  proj HOMO
   level -1
  end
  proj LUMO
   level 1
  end
  proj H-plus-L
   level from -1 to 1
  end
%endblock
```

TBT.Proj.<>.States **false** *(logical)*

Save all states for the projection. The saved quantity can be post-processed to decipher the locality of each projection.

Needed if you wish to select specific molecular orbitals dependent on the nature of the molecular orbital.

TBT.CDF.Proj.Compress **<TBT.CDF.Compress>** *(integer)*

Allows a different compression for the projection file. The projection file is typically larger than the default output file, so compression of them separately might be needed.

TBT.Projs.Init **false** *(logical)*

Whether TBTRANS will only create the projection tables and then quit.

As TBTRANS allows to re-use the projection file the user can choose to stop right after creation. Specifically it will allow one to swap projection states with other projection states. This can be useful when bias is applied and the hybridisation "destroys" the molecule Hamiltonian. After initialising the projection tables the user can manually swap them with those calculated at zero bias, thus retaining the same projection tables for different bias'.

Note that for spin calculations you need to utilise the **TBT.Spin** flag to initialise both projection files (spin UP *and* spin DOWN) before proceeding with the calculation.

TBT.Projs.Debug **false** *(logical)*

Print out additional information regarding the projections. It will print out assertion lines orthogonality.

Possibly not useful for other than the developers.

%block TBT.Projs.T ⟨**None**⟩ *(block)*

As you might specify *many* molecular projections to investigate a lot of details of the system it seems perilous to always calculate all allowed transmission permutations.

Instead the user has to supply the permutations of transport that is calculated. This block will let the user decide which to calculate and which to not.

In the following **Left**(L)/**Right**(R) corresponds to $T = \mathrm{Tr}[\mathbf{G}\mathbf{\Gamma}_L\mathbf{G}^\dagger\mathbf{\Gamma}_R]$ where **Left**, **Right** are found in the **TBT.Elecs** block.

from <proj-L> to Projects $\mathbf{\Gamma}_L$ on to the **<projection>** before doing the R projections.

The R projections are constructed in the following lines until **end** is seen.

<proj-R> Projects $\mathbf{\Gamma}_R$ on to the **<projection>** which then calculates the transmission

Each projection can be represented in three different ways:

<elec> Makes no projection on the scattering matrix

<elec>.<name> Makes all permutations of the projections attached to the molecule named **<name>**

<elec>.<name>.<P-name> Projects the named projection **<P-name>** from molecule **<name>** onto electrode **<elec>**

An example input for projection two molecules could be:

```
%block TBT.Projs.T
  from Left.M-L.HOMO to
    Right.M-R
    Right
  end
  from Left.M-L.LUMO to
    Right.M-R.LUMO
  end
%endblock
```

which will be equivalent to the more verbose

```
%block TBT.Projs.T
```

```
  from Left.M-L.HOMO to
    Right.M-R.HOMO
    Right.M-R.LUMO
    Right.M-R.H-plus-L
    Right
  end
  from Left.M-L.LUMO to
    Right.M-R.LUMO
  end
%endblock
```

This will calculate the transport using all these equations

$$T_{|H_1\rangle,|H_2\rangle} = \mathrm{Tr}\left[\mathbf{G}|H_1\rangle\langle H_1|\mathbf{\Gamma}_L|H_1\rangle\langle H_1|\mathbf{G}^\dagger|H_2\rangle\langle H_2|\mathbf{\Gamma}_R|H_2\rangle\langle H_2|\right] \tag{8}$$

$$T_{|H_1\rangle,|L_2\rangle} = \mathrm{Tr}\left[\mathbf{G}|H_1\rangle\langle H_1|\mathbf{\Gamma}_L|H_1\rangle\langle H_1|\mathbf{G}^\dagger|L_2\rangle\langle L_2|\mathbf{\Gamma}_R|L_2\rangle\langle L_2|\right] \tag{9}$$

$$T_{|H_1\rangle,|H_2\rangle+|L_2\rangle} = \mathrm{Tr}\left[\mathbf{G}|H_1\rangle\langle H_1|\mathbf{\Gamma}_L|H_1\rangle\langle H_1|\mathbf{G}^\dagger\big(|H_2\rangle\langle H_2| + |L_2\rangle\langle L_2|\big)\mathbf{\Gamma}_R\big(|H_2\rangle\langle H_2| + |L_2\rangle\langle L_2|\big)\right] \tag{10}$$

$$T_{|H_1\rangle,R} = \mathrm{Tr}\left[\mathbf{G}|H_1\rangle\langle H_1|\mathbf{\Gamma}_L|H_1\rangle\langle H_1|\mathbf{G}^\dagger\mathbf{\Gamma}_R\right] \tag{11}$$

$$T_{|L_1\rangle,|L_2\rangle} = \mathrm{Tr}\left[\mathbf{G}|L_1\rangle\langle L_1|\mathbf{\Gamma}_L|L_1\rangle\langle L_1|\mathbf{G}^\dagger|L_2\rangle\langle L_2|\mathbf{\Gamma}_R|L_2\rangle\langle L_2|\right] \tag{12}$$

Notice that 10 is equivalent to 11 in our two state model.

Note that removing an explicit named projection allows easy creation of all available permutations of the projection states associated with the molecule.

TBT.Projs.Only **false** *(logical)*

Whether TBtrans will not calculate non-projected transmissions. If you are only interested in the projection transmissions and/or have already calculated the non-projected transmissions you can use this option.

TBT.Projs.DOS.A **false** *(logical)*

Save the spectral density of states for the projections. In case you have set **TBT.DOS.A** this will default to that flag.

TBT.Projs.Current.Orb **false** *(logical)*

depends on: **TBT.Projs.DOS.A**

Will calculate and save the orbital current for the device with the projections.

The orbital current will be saved in the same sparsity pattern as the cut-out device region sparsity pattern.

TBT.Projs.T.All **false** *(logical)*

Same as **TBT.T.All**, but for projections. If differing projections are performed on the scattering states the transmission will not be reversible. You can turn on all projection operations using this flag.

TBT.Projs.T.Out **false** *(logical)*

Same as **TBT.T.Out** for projections.

4.6.3 NetCDF4 support

TBtrans stores all relevant physical quantities in the `SystemLabel.TBT.nc` file which retains orbital resolved DOS, orbital currents, transmissions, transmission eigenvalues, etc. One may use sisl to easily analyze and extract quantities from this file using Python.

These files are created if NetCDF4 support is enabled

`SystemLabel.TBT.nc` File which contain nearly everything calculated in TBtrans. The structure of this file is a natural tree structure to accommodate N electrode output.

`SystemLabel.TBT.SE.nc` see **TBT.CDF.SelfEnergy.Save**

Down-folded self-energies are stored in this file.

`SystemLabel.TBT.Proj.nc` see **TBT.Projs**

Stores projected DOS, transmission and/or orbital currents. Using projections for large k and energy sampling will create very large files. Ensure that you have a large amount of disk-space available.

`SystemLabel.DOS` see **TBT.DOS.Gf**

The k resolved density of states from the Green function.

`SystemLabel.AVDOS` see **TBT.DOS.Gf**

The k averaged density of states from the Green function.

`SystemLabel.ADOS_<>` see **TBT.DOS.A**

The k resolved density of states from the electrode name **<>**.

`SystemLabel.AVADOS_<>` see **TBT.DOS.A**

The k averaged density of states from the electrode name **<>**.

`SystemLabel.TRANS_<1>_<2>`

The k resolved transmission from **<1>** to **<2>**.

`SystemLabel.AVTRANS_<1>_<2>`

The k averaged transmission from **<1>** to **<2>**.

`SystemLabel.CORR_<1>` see **TBT.T.Out**

The k resolved correction to the transmission for **<1>**.

`SystemLabel.AVCORR_<1>` see **TBT.T.Out**

The k averaged correction to the transmission for **<1>**.

`SystemLabel.TEIG_<1>_<2>` see **TBT.T.Eig**

The k resolved transmission eigenvalues from **<1>** to **<2>**.

`SystemLabel.AVTEIG_<1>_<2>` see **TBT.T.Eig**

The k averaged transmission eigenvalues from **<1>** to **<2>**.

`SystemLabel.CEIG_<1>` see **TBT.T.Out**

The k resolved correction eigenvalues for **<1>**.

`SystemLabel.AVCEIG_<1>` see **TBT.T.Out**

The k averaged correction eigenvalues for **<1>**.

`SystemLabel.BDOS_<>` see **TBT.DOS.Elecs**/**TBT.T.Bulk**

The k resolved bulk density of states of electrode <>.

`SystemLabel.AVBDOS_<>` see **TBT.DOS.Elecs**/**TBT.T.Bulk**

The k averaged bulk density of states of electrode <>.

`SystemLabel.BTRANS_<>` see **TBT.DOS.Elecs**/**TBT.T.Bulk**

The k resolved bulk transmission of electrode <>.

`SystemLabel.AVBTRANS_<>` see **TBT.DOS.Elecs**/**TBT.T.Bulk**

The k averaged bulk transmission of electrode <>.

All the above files will only be created if TBtrans was successfully executed and their respective options was enabled.

4.6.4 No NetCDF4 support

In case TBtrans is not compiled with NetCDF4 support TBtrans is heavily limited in functionality and subsequent analysis. Particularly the DOS quantities are not orbital resolved. Also none of the quantities will be k averaged, this is required to be done externally.

The following files are created:

`SystemLabel.DOS` see **TBT.DOS.Gf**

The k resolved density of states from the Green function.

`SystemLabel.ADOS_<>` see **TBT.DOS.A**

The k resolved density of states from the electrode name <>.

`SystemLabel.TRANS_<1>_<2>`

The k resolved transmission from **<1>** to **<2>**.

`SystemLabel.TEIG_<1>_<2>` see **TBT.T.Eig**

The k resolved transmission eigenvalues from **<1>** to **<2>**.

`SystemLabel.BDOS_<>` see **TBT.DOS.Elecs**/**TBT.T.Bulk**

The k resolved bulk density of states of electrode <>.

`SystemLabel.BTRANS_<>` see **TBT.DOS.Elecs**/**TBT.T.Bulk**

The k resolved bulk transmission of electrode <>.