ANT-1D

The Alicante Nano Transport program

David Jacob
MPI für Mikrostrukturphysik
Weinberg 2, 06108 Halle, GERMANY
Email: djacob@mpi-halle.de

August 6, 2010

1 Introduction

ANT-1D is a program for calculating the transport properties of nanosystems starting from the (effective) one-body Hamiltonian of the system represented in a tight-biding form or in some local basis set. The transport properties are computed in the framework of the Landauer formalism in connection with the one-body Green's function method. The electrodes are quasi one-dimensional nanowires with a finite cross section.

2 Method

A detailed account of the methodology can be found elsewhere, for example in Refs. [1, 2, 3, 4, 5, 6], and references therein.

3 Functionality

Basic functionality:

- Calculation of Landauer transmission function
- Calculation of DOS and projected DOS
- Population analysis

- Determination of Charge and Fermi level
- Interface with **CRYSTAL06** *ab initio* electronic structure code

Advanced functionality for treating strong electronic correlations: [5]

- Calculation of hybridization functions to calculate electronic self-energy of strongly interacting electrons with impurity solvers
- Correlated transmission function
- Correlated DOS and PDOS
- Correlated charge and density matrix

4 Installation

The program is written in standard FORTRAN90. Compilation should thus be straight forward in principle. So far the program has been tested on Linux PCs (both AMD and INTEL) with the Portland Group Fortran compiler (**pgf90**) and the INTEL FORTRAN compiler (**ifort**).

To compile the **ANT-1D** code, go to the subdirectory src and edit the Makefile: You have to set the variable F90 to the FORTRAN90 compiler you want to use to compile the program. In the variable EXTRALIBS additional libraries that have to be linked to generate the executable, have to be specified together with the path where the compiler can find them. The only external libraries needed are the BLAS/LAPACK libraries. But some compilers (like **ifort**) need additional libraries to link these libraries. Please see the documentation of your compiler package.

The **ANT-1D** code has been parallized with MPI. In order to compile with MPI, the `-DHAVE_MPI` flag has to be activated in the Makefile. Furthermore the flag `F90` has to be set to mpif90. The parallel code has been succesfully tested with **openmpi** and **mpich**.

5 Running ANT

ANT-1D takes input from a single input file. The input file must have the extension `.ant`. When invoking Alitrans the input file (`<input>.ant`) has to be specified as a command line parameter without the extenison:

```
ant2010 <input>
```

or in order to redirect the output to a file.

```
ant2010 <input>  >& <output>
```

If **ANT-1D** has been compiled with MPI (see above), then one can run the code in parallel with the following command:

```
mpirun -np <N> ant2010 <input>
```

or redirecting the output:

```
mpirun -np <N> ant2010 <input> >& <output>
```

where `<N>` is the number of proecessors to be used.

5.1 Input files

Normally, five input files are necessary to run ANT:

1. the *main input file* (extension `.ant`) contains the definitions of various parameters and specifies the names of the other input files;
2. the *device input file* (extension `.dev`) defines some parameters specific to the device part and most importantly the device Hamiltonian and overlap matrix;
3. the *device geometry file* (extension `.xyz`) specifies the atomic geometry of the device
4. the two *electrode input files* (extension `.elc`) define parameters specific to the electrode parts and the electrode Hamiltonian and overlap matrix;

Each file consists of one or several input sections in the FORTRAN90 Namelist format. A namelist specification starts with an ampersand and the name of the Namelist. The namelist specification is ended with "/". Values to the namelist parameters can be assigned by equating the parameter's name to an allowed value. Parameters within the namelist can be specified in any

order, and even repeatedly. If parameters are specified repeatedly, the parameter holds the value of the last assignment. Specification of any namelist parameter is optional, i.e. the namelist can be empty. The parameters then hold default values. However, the namelist must be specified even if it is empty. The parameter list can be commented by using the usual Fortran90 comments, e.g. the exclamation mark "!". The following example illustrates the specification of some parameters in the general parameters section of the main input file:

```
&Parameters            ! Begin of namelist ‘‘Parameters’’
Lead1File = lead1.elc  ! Input file for lead #1
Lead2File = lead2.elc  ! Input file for lead #2
DevFile = device.dev   ! Input file for device
Transm = .true.        ! Compute Transmission and DOS
AtomDOS = 30, 31       ! Compute PDOS projected on orbtials
                       !   of atoms #30 and #31
FindEFD = 2            ! Search for device Fermi level using
                       !   Muller method
Pop = 1                ! Mulliken population analysis
L1DMaxCyc = 10         ! Maximum number of cycles in self-consistent
                       !   solution of Dyson equation for calculating
                       !   lead self-energies
Gamma = 1.0d-3         ! Broadening on real energy axis for computing
                       !   Transmission and DOS
ChargeAcc = 1.0d-2     ! Accuracy for computing charge
FermiAcc = 1.0d-2      ! Accuracy for Fermi level search
/                      ! End of namelist ‘‘Parameters’’
```

Note, that by the FORTRAN90/95 standard the input is *not* case sensitive. For furhter details of the namelist construct, see the FORTRAN90/95 documentation.

Additionally, other important and mandatory parameters like the Hamlitonians and overlap matrices of device and leads are specified outside the namelist constructs in a special format. In the following we explain the input format, and give a complete list of the parameters that can be specified in each of the input files.

5.2 Main input file

The main input file can have any name but it *must* end with the extension `.ant`. It contains the mandatory `Parameters` section which specifies various general parameters. Additionaly, a number of `AtomData` namelists must be

specified depending on the value of the variable `NAtomData` in the `Parameters` namelist. Below we list the names of the variables that can be altered within the `Parameters` namelist, together with the default value (in parenthesis), and a short explanation of the variable's function.

General parameters

`ChargeAcc` (`Real`, default: `1.0E-03`):
Absolute accuracy when calulating the charge, e.g. the number of electrons (of device or leads) by integrating Greens functions.

`FermiAcc` (`Real`, default: `1.0E-03`):
Absolute accuracy when calculating Fermi level (of device or leads) using `FindEFL` or `FindEFD` options.

`eta` (`Real`, default: `1.0E-10`):
Infinitesimal imaginary part added to real energy when calculating the Greens function on the real axis.

`Gamma` (`Real`, default: `1.0E-03`):
Additioanl imaginary part added to real energy when calulating the electrode self-energies on the real axis in order to achieve a smoother surface DOS of the electrodes.

`infty` (`Real`, default: `100`):
This parameter determines the low energy cutoff (=`-infty`) for the charge integration.

`L1DMaxCyc` (`Integer`, default: `1000`):
Maximum number of cycles in self-consistent solution of Dyson equation for lead self energies.

`L1DConv` (`Real`, default: `1.0E-006`):
Convergence criterion for self-consistent solution of Dyson equation of lead self energies.

`L1Dalpha` (`Real`, default: `0.5`):
Mixing of self-energies in self-consistent solution of Dyson equation for lead self energies.

`NAtomData` (`Integer`, default: `0`):
Number of atomic data input blocks. See below.

Functionality parameters

`FindEFL` (`Integer`, default: `0`):
Whether to search for the Fermi levels of the leads.

`FindEFD` (`Integer`, default: `0`):
Whether to search for the Fermi level of the device. 0: Do not search for Fermi level; 1: Use Bisection method; 2: Use Müller method; 3: Use Secant method.

`DeltaEF` (`Real`, default: `0.5`):
Defines the initial energy window (EF-DeltaEF, EF+DeltaEF) for Fermi level search. Note that for the Bisection method (FindEFD=1) the energy must lie within these boundaries.

`PrintHS` (`Logical`, default: `.false.`):
Print out Hamiltonian and overlap matrices.

`ShrDev` (`Logical`, default: `.false.`):
Shrinks central device region by one lead unit cell on each side of the device.

`ExtDev` (`Logical`, default: `.false.`):
Extends central device region by one lead unit cell on each side of the device.

`E1` (`Real`, default: `-5.0`):
Lower bound of energy window for output of transmission and DOS.

`E2` (`Real`, default: `5.0`):
Upper bound of energy window for output of transmission and DOS.

`NPoints` (`Integer`, default: `1000`):
Number of energy points for which to calculate transmission and DOS.

`Pop` (`Integer`, default: `0`):
Whether to perform a population analysis of the device region. The population of each atom in the central device region is printed out. 0: No population analysis; 1: Mulliken population analysis; 2: Löwdin population analysis; 3: Direct population analysis (not taking into account overlap between atomic orbitals).

`AtomPop` (`Integer` array, default: `1000*0`):
For which atoms to perform a more detailed population analysis resolved in individual atomic orbitals. Assign an integer array of all possible atoms to AtomPop.

`Transm` (`Logical`, default: `.false.`):
Whether to calculate the transmission, DOS, and local DOS.

`AtomDOS` (`Integer` array, default: `1000*0`):
For which atoms to calculate a local DOS, e.g. resolved for individual atomic orbitals.

`AODOS` (`Integer` array, default: `1000*0`):
Specify individual atomic orbitals for which to calculate local DOS.

`NChannels` (`Integer`, default: `0`):
Print out transmissions of eigen channels.

`LeadDOS` (`Logical`, default: `.false.`):
Whether to print out the DOS of the bulk leads.

Parameters determining input file names:

`DevFile` (`Character` string, default: `device.dev`):
Specify file name of device input file.

`Lead1File` (`Character` string, default: `lead1.elc`):
Specify file name of left (1) lead input file.

`Lead2File` (`Character` string, default: `lead2.elc`):
Specify file name of right (2) lead input file.

`DevXYZ` (`Character` string, default: `device.xyz`):
Specify file name of device geometry input file.

`Lead1XYZ` (`Character` string, default: `lead1.xyz`):
Specify file name of geometry input file for lead no. 1.

`Lead2XYZ` (`Character` string, default: `lead2.xyz`):
Specify file name of geometry input file for lead no. 2.

Parameters for Electron Correlations:

`NCorrBl` (`Integer`, default: `0`):
Specifies the number of correlated blocks in the device region. Each correlated block can consist of several consequitve orbitals.

`CorrBl` Array(1:100) of type `TCorrBlock`:
Each array element specifies the parameters for each of the correlated blocks in the device region.

`TCorrBlock` elements:

`%beg` (`Integer`, default: `0`):
Number of 1st orbital of correlated block

`%end` (`Integer`, default: `0`):
Number of last orbital of correlated block

`%U` (`Real`, default: `0.0`):
On-site Coulomb repulsion (only used for double-counting correction)

`%J` (`Real`, default: `0.0`):
Hund's rule coupling (only used for double-counting correction)

`%nu` (`Real`, default: `0.0`):
Total occupation of spin-up orbitals in correlated block (only used for double-counting correction)

`%nd` (`Real`, default: `0.0`):
Total occupation of spin-down orbitals in correlated block (only used for double-counting correction)

`Correlated` (`Logical`, default: `.false.`):
If activated all calculations are performed taking into account electronic correlations described by a dynamic self-energy for all correlated orbitals. For each correlated block (see. `CorrBl` above) the self-energies are read from a single file `sig.inp.<iblock>` where `<iblock>` is the number (=`1 - <NCorrBl>`) of the corresponding correlated block. For each orbital within the correlated block, two columns corresponding to the real and the imaginary part of the self-energy have to be specified.

`HybFunc` (`Logical`, default: `.false.`):
This option allows to calculate the hybridization function for each of the correlated blocks defined by the parameters `NCorrBl` and `CorrBl`. The hybridization function is one of the important quantities that define the Anderson impurity problem, and hence is a necessary input for the impurity solver. If the option `Correlated` has been activated the energy mesh on which to calculate the hybridization function is defined by the energy mesh of the self-energy input files `sig.inp.<iblock>`. Otherwise the mesh has to be defined in an additional file `mesh.dat`.

`DiagCorrBl` (`Logical`, default: `.false.`):
If this option is activated the single-particle Hamiltonian of the correlated block is diagonalized (and previously orthogonalized). This is useful when the strongly interacting electrons are described by more than one set of orbtials.

For example in a certain basis set, the strongly interacting $3d$-electrons could be described within the ab initio calculation by more than one set of d-orbitals. The diagonalization then effectively contracts the d-shells into a single low-energy d-shell which si then correlated.

`SymmCorrBl` (`Logical`, default: `.false.`):
This option averages the on-site energies of the orbitals within a correlated block. This can be useful if the *ab initio* calculatuion has broken the symmetry of the correlated states too strongly.

AtomData input blocks

Additionaly, atomic data input blocks must be specified when `NAtomData > 0`. The atomic data input blocks contain information about the atomic types (identified by their atomic number) used in the calculations. There must be as many atomic data input blocks as specified by the parameter `NAtomData` in the Parameters namelist. The individual atomic data blocks are again Fortran90 namelist format, and the namelist name is `AtomData`. If no atomic data is specified (`NAtomData = 0`) some of the above options of `Parameters` namelist which require atomic information (e.g. `AtomPop, AtomDOS` cannot be used.

At the moment there are only two variables that can be modified in each atom data block:

`AN` (`Integer`, default `0`):
Atomic number of atom type.

`AtShells` (`Character` array):
Array of characters specifying atomic shells of atom: s,p,d,f.

Example of main input file:

```
&Parameters
Lead1File = lead1.elc
Lead2File = lead2.elc
DevFile = device.dev
FindEFD = 0
DeltaEF = 0.1
Pop = 1
AtomPop = 30, 31
Transm = .true.
AtomDOS = 30, 31
Shrdev = .true.
```

```
L1DMaxCyc = 10
Gamma = 1.0d-3
ChargeAcc = 0.001
FermiAcc = 0.001
NAtomData = 2
/

&AtomData
AN = 227
AtShells = s p d
/

&AtomData
AN = 229
AtShells = s p d
/
```

5.3 Device input file

The device input file should have the extension `.dev`, It contains the definition of the device Hamiltonian and overlap matrices, and the specification of some parameters. The device parameters are specified in the namelist `DevParam` at the beginning of the device input file:

```
&DevParams
NDSpin = <integer> ! spin-degenerate (=1,default) or spin-polarized (=2)
NDAO =   <integer> ! Number of atomic orbitals in device region
NDEl =   <integer> ! Number of electrons in device region
EFermi = <real>    ! Fermi level (default = 0.0)
sparse = <logical> ! whether to read Hamiltonian and overlap matrices
                   ! as sparse matrices (=.true.), or not (=.false.,default)
/
```

Subsequently, the Hamiltonian matrix and the overlap matrix of the device region are specified. In the case of a spin-polarized calculation (`NDSpin=2`) a Hamiltonian matrix for each spin has to be specified. For non-sparse mode (`sparse=.false.`), the whole matrix has to be specified line by line. For the case of 3 orbitals and spin-degenerate calculation we would have for example:

```
! Hamiltonian
 0.0 -1.0  0.0
-1.0  0.0 -1.0
```

```
 0.0 -1.0  0.0

! Overlap
 1.0  0.1  0.0
 0.1  1.0  0.1
 0.0  0.1  1.0
```

Note that lines starting with an exclamation mark are comment lines and are ignored. In the case of an orthogonal basis set (i.e. the overlap matrix is the matrix identity), one does not have to specify the overlap matrix if the keyword `OrthogonalBS` in the general `Parameters` namelist is set `.true.`

For sparse-matrix mode (`sparse=.true.`) only the non-zero matrix elements have to be specified. The non-zero matrix elements are specified by the matrix indices i,j and the corrsponding matrix elements A_{ij}. The end of the sparse matrix input is marked by the key `0 0 0.0`. The above example in sparse-matrix mode would look like:

```
! Hamiltonian
1 2  -1.0
2 1  -1.0
2 3  -1.0
3 2  -1.0
0 0 0.0

! Overlap
1 1  1.0
2 2  1.0
3 3  1.0
1 2  0.1
2 1  0.1
2 3  0.1
3 2  0.1
0 0 0.0
```

5.4 Electrode input file

The electrode input file(s) (extension `.elc`) are very similar to the device input file described in the previous section. At the beginning of the input files one has to specify certain parameters inside the `LeadParams` Namelist:

```
&LeadParams
NSpin  = <integer> ! 1 = spin-degenerate, 2 = spin-polarized
```

```
NPC    = <integer> ! Order of neighbour hoppings between
                   ! primitive cells to be taken into account
NPCAO  = <integer> ! Number of atomic orbitals in primitive cell
NPCEL  = <integer> ! Number of electrons in primotive cell
EFermi = <real>    ! Fermi level of electrodes
sparse = <logical> ! whether to read Hamiltonian and overlap
                   ! matrices as sparse
/
```

Next, the Hamiltonian and overlap matrices within the primitive cells and between primitive cells have to be specified. The variable `NPC` specified in the above `LeadParams` Namelist gives the number of inter-cell hoppings and overlaps to be taken into account, i.e. for `NPC=1` only nearest neighbour hoppings and overlaps are taken into account, for `NPC=2` also next-nearest neighbour hoppings and overlaps are taken into account. Note that each primitive cell in the electrodes can contain more than one atom.

As in the Device input file, first the Hamiltonians have to be specified, then the overlap matrices. In the case of an orthogonal basis set (i.e. the overlap matrix is the matrix identity), one does not have to specify the overlap matrix if the keyword `OrthogonalBS` in the general `Parameters` namelist is set `.true.`

For the case of 1 orbital in the primitive cell, nearest-neighbour hopping (`NPC=1`) and a spin-degenerate calculation we would have for example:

```
! v0 (intra-cell Hamiltonian)
 0.0

! v1 (nearest neighbour interaction)
-1.0

! s0 (intra-cell Overlap)
 1.0

! s1 (nearest neighbour Overlap)
0.1
```

If the `sparse` switch has been set, each matrix has to be specified in the sparse matrix format as explained in the section above for device input.

5.5 Device and electrode geometry files

In the Device and Electrodes geometry files the atomic geometries of the device region and the electrode primitive cells can be specified. The format

is the usual xyz-file format:

```
<NAtoms>

<AN1> <x1> <y1> <z1>
<AN2> <x2> <y2> <z2>
  :     :    :    :
  :     :    :    :
<ANN> <xN> <yN> <zN>
```

`<AN1>, <AN2>...` `<ANN>` refer to the atomic numbers of the 1st, 2nd, and n-th atom. `<x1>` `<y1>` `<z1>` are the coordinates of the 1st atom, and so on. The atomic number there must be a corresponding `AtomData` Namelist in the main input data file.

Note that the geometry does not have to be specified. In that case on can simply set `<NAtoms>` to zero in the geometry file. However, in that case it is not possible to use some of the options of the `Parameters` Namelist in the main input file requiring informations about the atoms, as for example `AtomDOS`, `AtomPop` etc.

5.6 Output files

General information during the processing of the input files and the calculation is written to the standard output and can be redirected to a file via the `>` operator. On the other hand data is written to special output files.

Transmission data file

The transmission data file `<file>.transm.dat` contains the calculated transmission function as a function of energy if a calculation of the transmission has been requested in the main input file. The first column contains the energy E while the second column contains the corresponding transmission function $T(E)$. In the case of a spin-polarized calculation (`NDSpin=2`), the second column contains the transmission for the spin-up channel $T_\uparrow(E)$ and the third column contains the transmission for the spin-down channel $T_\downarrow(E)$. Note that the transmission is always understood as a transmission per spin-channel. Hence in the case of a not spin-polarized calculation (`NDSpin=1`) the total transmission is given by $2 \times T(E)$, and accordingly the conductance is $\frac{2e^2}{h} \times T(E)$. In the spin-polarized case on the other hand, the total conductance is $\frac{e}{h} \times (T_\uparrow(E) + T_\downarrow(E))$. If eigenchannel are to be calculated (keyword `NChannels`) the following columns contain the individual eigenchannel

transmissions in descending order. For spin-polarized case, first all spin-up eigenchannels, then all spin-down eigenchannels are printed out.

DOS data file

This file (`<file>.dos.dat`) is always written when a transmission calculation is requested. It contains the total density of states (DOS) of the device region and (if requested) the projected DOS for all the atomic orbitals of the atoms specified by the `AtomDOS` array. The format is similar to the transmission data file: The first column contains the energy E, and the second column contains the corresponding total DOS of the device region $\rho_{\mathrm{D}}(E)$. For a spin-polarized calculation, the spin-resolved total DOS of the device region is written into the second (spin-up) and third (spin-down) column. The following columns contain the projected DOS of each atomic orbital as specified by the `AtomDOS` array. The order is the following: First, all the atomic orbitals of the first atom specified by the `AtomDOS` array are printed out, then all the atomic orbitals of the second atom and so forth. For the spin-polarized case, first all projected DOS for spin-up, and then all projected DOS for spin-down are printed.

5.7 Interface to CRYSTAL: cry2ant

With the cry2ant interface one can generate **ANT-1D** input from the output of a converged **CRYSTAL06** electronic structure calculatuion of a 1D-system (keyword `POLYMER` in CRYSTAL). In order to do so CRYSTAL must have written the Kohn-Sham Hamiltonian and overlap matrices to the CRYSTAL output file (`<file>.out`). This is achieved by putting the directive `PRINT` at the end of the basis set input section in the CRYSTAL input file (`<file>.d12`) and specifying the following lines in the CRYSTAL properties input file (`<file>.d3`):

```
BASISSET
2
60 <n>
64 <n>
END
```

where the `<n>` stands for an integer number. It specifies the number of nearest neighbour interactions to write to the output file: For $n = 1$ only the intra unit-cell Hamiltonian is written to the output. this is enough when only device input is generated from the output. If input for the leads is to

be generated, one has to output at least the interactions with the two neighbouring cells, i.e. $n = 3$. For output of next-nearest neighbour interactions take $n = 5$, and so on (i.e. n alwasy has to be uneven).

Once you have the CRYSTAL06 output files with the Kohn-Sham Hamiltonian and overlap matrices, you have to run the interface program `cry2ant` in the shell. The command line is:

```
cry2ali <CRYSTAL output> [options]
```

The first (mandatory) input parameter is the the CRYSTAL output file. By default (i.e. without further specifications of options) it will generate lead input files taking as many inter-cell hoppings as found in the CRYSTAL output.

The following options can be specified to control the generation of **ANT-1D** input files:

1. `-r` : **Reorder Atoms**
   This options reorders the atoms within the unit-cell according to their coordinates. The use of this option is almost *always necessary* in order to connect the device region correctly with the leads when running the ANT-1D program.

2. `-z` : **Rotate to z-axis**
   The crystal code aligns one-dimensional systems with the x-axis. However, it is often more convenient to align the transport direction with the z-direction. This option rotates the coordinate system so that the transport direction is along the z-axis.

3. `-d` : **Generate device input**
   This option must be used if device input files are to be generated.

4. `-npc <N>` : **Take up to N-th nearest neighbour hoppings**
   Only active for generation of lead input files. This option restricts the number of nearest neighbour hoppings to N.

5. `-cut <N1> <N2>` : **Cut out device from CRYSTAL unit cell** Only acitve for generation of device input file. This option allows to cut out a smaller region from the CRYSTAL unit cell. $N1$ and $N2$ are positive integers specifying the first atom and the last atom of the unit cell that are taken as the device region.

6 Examples

In the 'examples' subdirectory a number of simple examples can be found that illustrate the functionality of `ANT-1D`.

References

[1] J. J. Palacios *et al.*, Phys. Rev. B **66**, 035322 (2002).

[2] D. Jacob, J. Fernández-Rossier and J. J. Palacios, Phys. Rev. B **73**, 075429 (2006).

[3] D. Jacob, PhD thesis 2007, Universidad de Alicante; arXiv:0712.1383.

[4] D. Jacob, J. Fernández-Rossier and J. J. Palacios, Phys. Rev. B **77**, 165412 (2006).

[5] D. Jacob, K. Haule and G. Kotliar, Phys. Rev. Lett. **103**, 016803 (2009).

[6] D. Jacob and J. J. Palacios, in preparation (2010).