User Manual for KKRSUSC

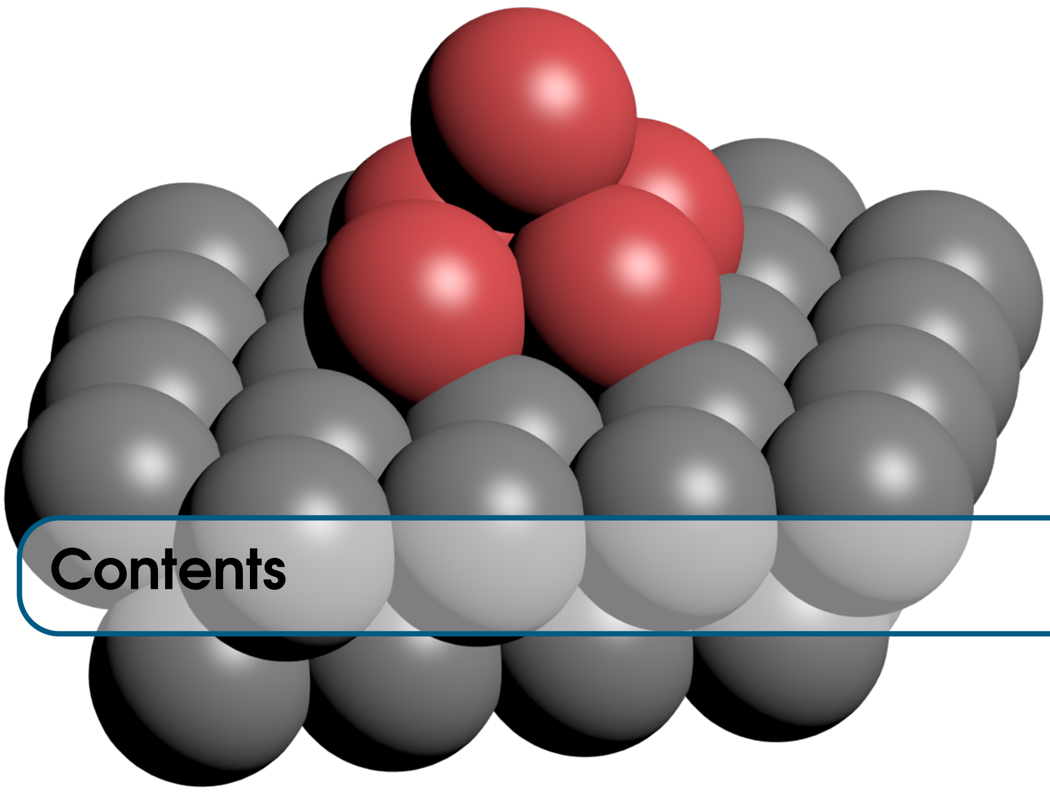
Linear response KKR package

Julen Ibañez, Jonathan Chico, Manuel Dos Santos *et al.*

PUBLISHED BY PUBLISHER

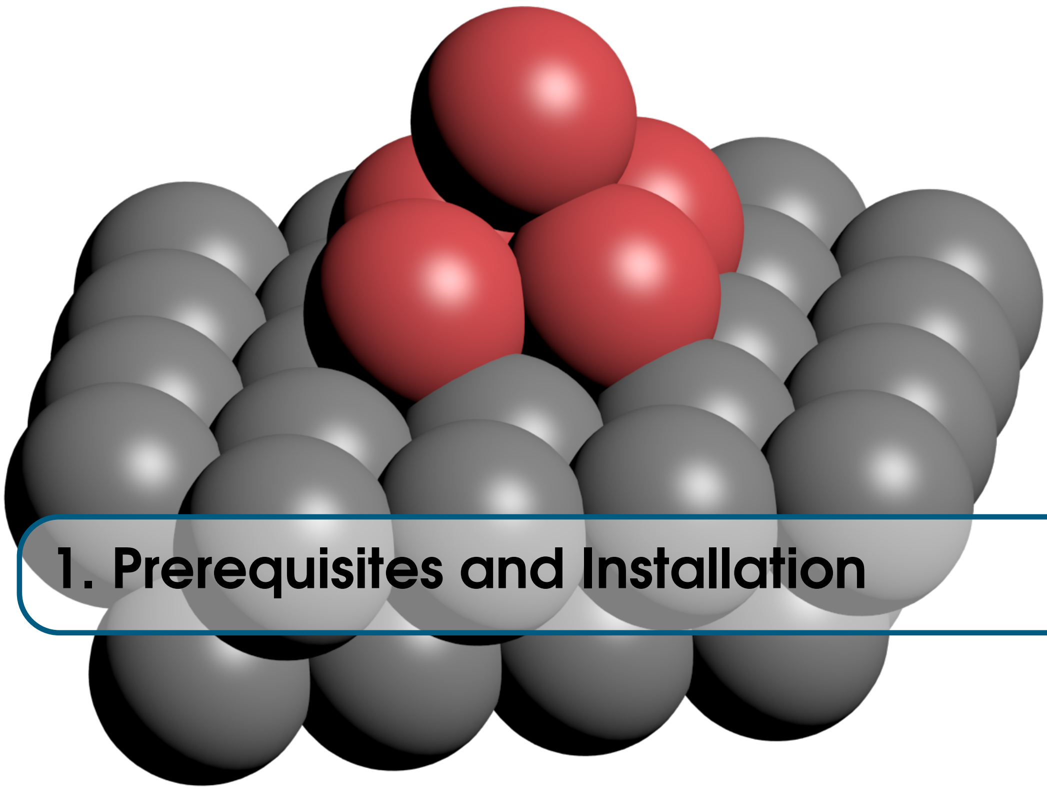
BOOK-WEBSITE.COM

First printing, June 2017



Contents

Part One



1. Prerequisites and Installation

The KKRSUSC program is an add-on to the main Jülich-Münich KKR JM-KKR *ab-initio* software package and its extension for impurities KKRFLEX, that allows the calculation of dynamical properties of the system via linear response theory. The user is strongly encougraged to take some time and read the manuals and or wikis to both those packages, as a complete description of the cabailities, pitfalls and strategies for each of those programs is beyond the scope of this manual. However, basic information about each of those programs will be provided as the information that they require is necessary to be able to properly use the KKRSUSC.

1.1 Getting the codes

The first step is to obtain the source code for the JM-KKR, KKRFLEX and KKRSUSC. All of these software packages are available trough the IFFGit service, to gain access to these projects contact one of the persons responsible for the code.

Why Git?

Git is a very powerful version control system with an excellent web interface, allowing the users to:

- Keep up to date with any changes/bugfixes in the main versions of the code.
- Allowing for a safe development environment where changes can be reversed in case of mistakes.
- Letting other users access the new implemented features.
- It counts with a forum like environment for developers so that any pressing issued and or bug reports can be easily addressed by the developers.

Once access has been granted to the Git service, and the user has properly configured git in their local machine (see the Git guide provided by iffgit), the user should create a clone of the repository, for this

```
git clone project@iffgit
```

of this way a local version of the code which is a copy of the **current** version of the code will be created. The user would then be able via the diverse git commannds (`git pull`, `git push` and `git merge`) to update their local versions with the most recent version of the code in the repository, or to push their latest changes to the repository.

It is recommended (or is it necessary?) for the user to have the intel fortran compiler (`ifort`), the MKL libraries and the intel MPI compiler. The different codes have different degrees of parallelism (Maybe ask Juba or Sascha to describe this) which can be used to accelerate the calculations.

1.2 Preparation to run

Before starting any calculation and or compiling any of the codes one must first see what type of calculation one wants to perform. Currently there are two types of solvers for the `KKRFLEX` code and the `KKRSUSC` code, whcih moostly differ in the way in which the spin orbit coupling (SOC) is terated.

The *older* version of the code, only allows the treatment of SOC as a perturbation in the `KKRSUSC`. And a *new* version which allows one to consider SOC all the way from the bulk calculation. Both calculations require that one setup and compile the `JM-KKR` code in a different way. Hence it is very important to know which version one wants to run, for a simple guide one can take a look at Fig. 1.1.

In general the procedure to be able to run the `KKRSUSC` calculation is the following, decide wheter or not one needs SOC in the host, if one does not prepare the `JM-KKR` for an ASA no SOC calculation (how to do this will be explained in the next chapter), if one needs SOC in the host, prepare the `JM-KKR` for a Full potential+SOC calculation. One this has been done perform an self-consitent calculation (SCF) of the potential of the host.

After that, prepare the needed files for the `KKRFLEX` (for details on that see chapter **??**), if running with SOC use the *new* solver, if not use the *old* solver. Perform the SCF calculation of the cluster system that one wishes to treat. After that if SOC has been cosndiered one can then calculate the linear response funcitons that one wishes with `KKRSUSC` without problem. If SOC has not been considered, run an SCF calculation+SOC in `KKRSUSC` using the converged (no SOC) potential from `KKRFLEX`, afterwards use that potential (`KKRSUSC`+SOC SCF) to calculate the quanitites of interest via linear response.

It is important to be consistent with the approaches used (shape of the potential, inclusion of SOC) to ensure the reliability of the results. If a wrong input it given sometimes the programs will realize and will print out warnings, however, to ensure loss of time and erroneous results it falls to the user to be careful with the I/O chain from `JM-KKR` all the way to `KKRSUSC`.

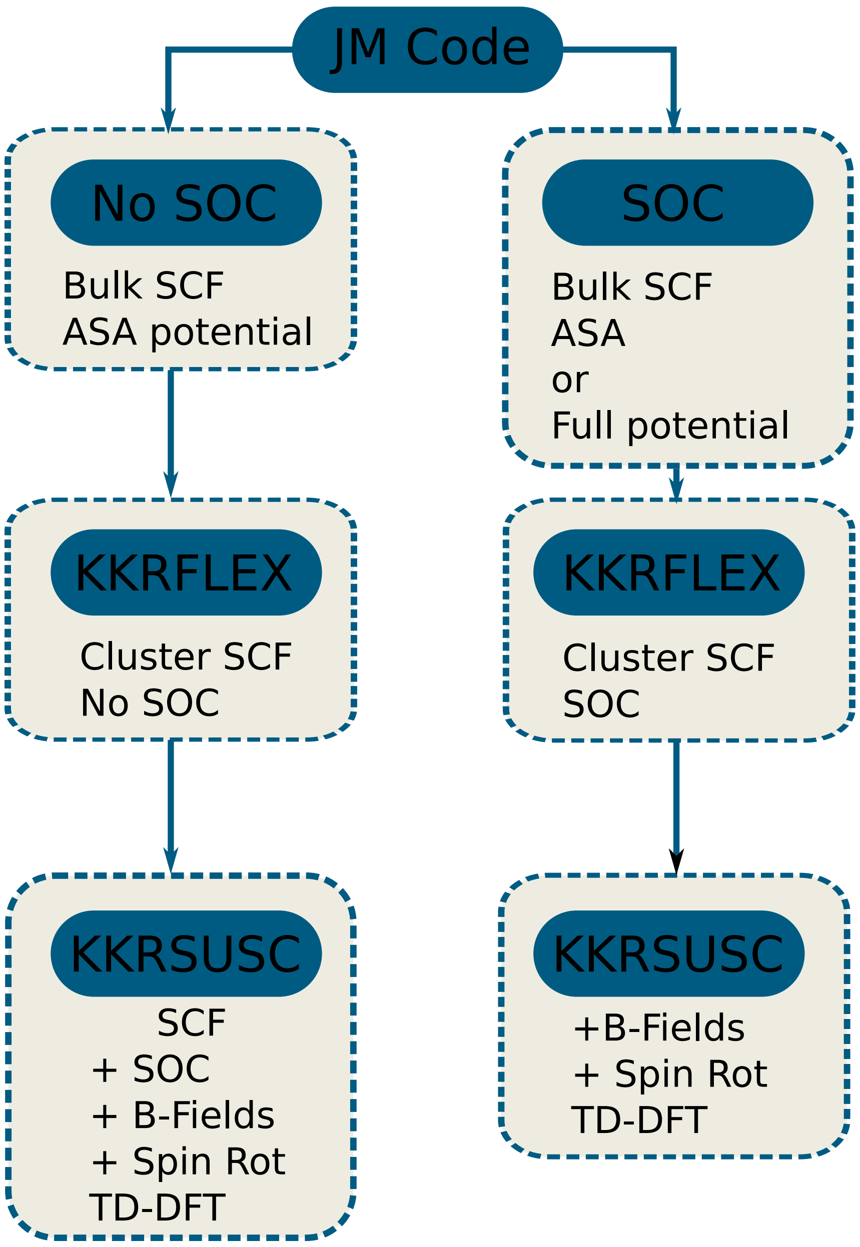
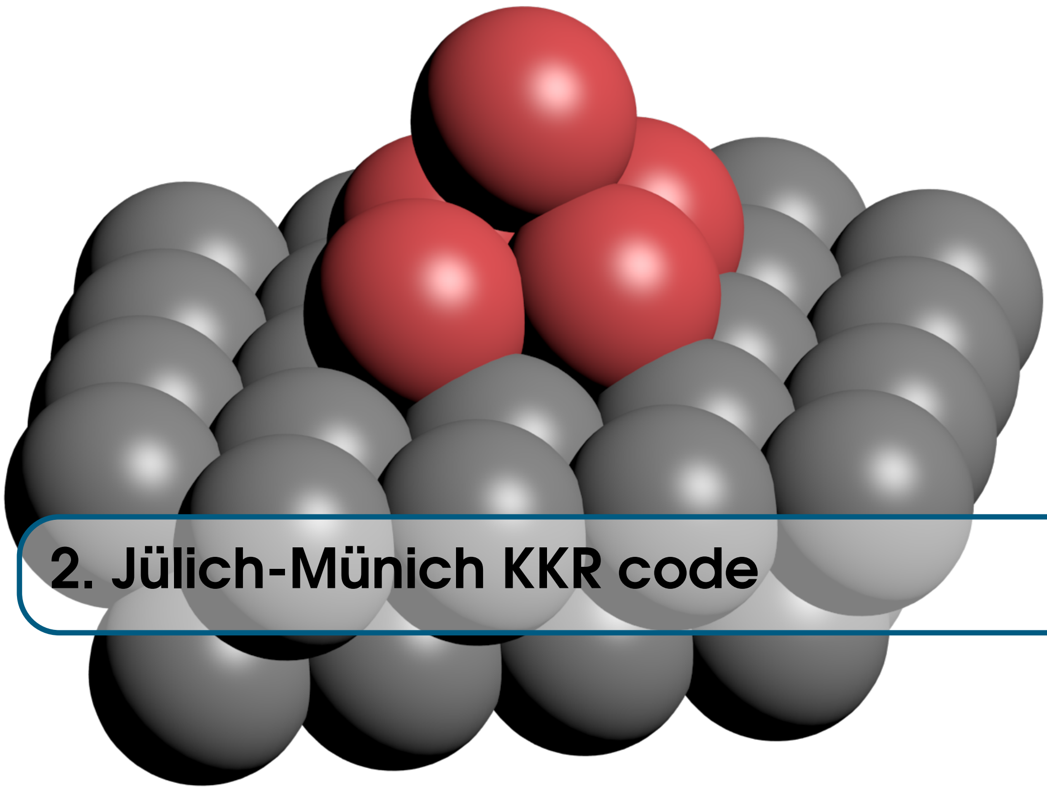


Figure 1.1: Strategy for the usage of the KKR software packages to run `KKRSUSC`.



2. Jülich-Münich KKR code

As mentioned in the previous chapter, to be able to perform linear response calculations in the current version of `KKRSUSC` one needs to first create the appropiate files resulting from previous calculations from other KKR codes. The first one to be explored is the Jülich-Münich KKR software package (`JM-KKR`). This is a firts principles calculations software package that uses the Korringa-Kohn-Rostocker (KKR) [**KKR**] approach to Density Functional Theory (DFT).

What is KKR?

The **Kohn-Korringa-Rostocker** (KKR) method is first principles calculations method based in **Multiple Scattering Theory** (MST), and usually making use of **Green functions**, that recasts the problem of an electron interacting with an effective potential in a solid as a multiple scattering problem.

The KKR method is often used for its capacity to make use of the **Dyson Equation**, which allows the inclusion of perturbation in a natural and simple way, thus making it a great formalism to study linear-response quantities.

This code allows for the calculation of many electronic and magnetic properties of a wide variety of systems, letting the user treat both bulk and slab structures, i.e. allowing the study of surfaces. For a full account of the capabilities of the `JM-KKR` code the user is encouraged to read the user guide and manual specific for that code. If well in this guide basic user knowledge of the `JM-KKR` will be given, it will be focused mostly on how it is used to generate the needed files for future calculations based on `KKRFLEX` and `KKRSUSC`.

2.1 Compile the `JM-KKR` program

Before compiling the `JM-KKR` code one must think about the type of system that one wants to simulate. This is because there are a set of arrays in the code with fixed lenght, which depend on the properties of the system. Some of the most common quantities that will-force the user to re-compile the code are the following:

- ASA or FP calculation. `KSHAPE`

- SOC or no SOC in the host. `KORBIT`
- Number of atoms in the unit cell. `NAEZD`
- Magnetic/non-magnetic host calculation. `KSP`
- Number of principal layers needed. `NPRINCD`
- Maximum value of the expansion for the angular momentum. `LMAXD`

As can be seen each of these variables has a well defined keyword, these can be found in a filne named `inc.p` which is found in the source folder of the `JM-KKR` code. If one is going to perform a calculation that requires one to change one or more of these variables **one must re-compile the code**.

After one has looked into the inc.p file and has made sure that all the variables presented there have the correct variables one must compile the code, to do that one must do the following in the source folder

- To compile the serial version:

```
make all
```

- To compile the parallel version:

```
make allmpi
```

- To clean:

```
make clean
```

Tip: Always be saving

One would probably need to run several simulations with the same number of atoms, magnetic/or non-magnetic host, shape of the potential, etc. Hence, it is a good idea to create a folder for each set of variables, where one can store both the executable `kkr.x` and the `inc.p` file. Of this way these files can be re-used and one would just need to modify the *flexible* variables in the input file of the code.

After saving the `inc.p` and the `kkr.x` files in the folders the code is ready to run. It is important to notice that there are many versions of the code, and the methods described here are valid for the *current* version of the code, i.e. any version from at least 05-01-2017.

2.1.1 Brief list of what each executable does for *legacy* versions of `JM-KKR`

In this version the `JM-KKR` has only one executable for the SCF run, however, there are some other versions of the code, which have several executables. These *legacy* versions of the code require the same care to the `inc.p` files as the *current* version. However, for their compilation one must run the `compile` script

```
./compile
```

This will compile these *legacy* versions of the code, and produce a series of executables, all of which perform different tasks:

- `kkr0.exe`
  - Reads the input files.
  - Looks at the geometry of the system.
- `kkr1a.exe`
  - Calculates the screened Green function.
- `kkr1b.exe`

  - Integrates over the energy loop.
  - Construct multiple scattering solutions.
- kkr1c.exe
  - Put together charge density from the Green functions.
- kkr2.exe
  - Calculate the total energy.
  - Also mix the potential.

2.2 The inputcard file

To be able to run the JM-KKR software one requires a series of files, in this section the inputcard file will be described. The inputcard is the main input file to run the JM-KKR code, where all the main options and auxiliarly files are described.

Lets see how a sample of an inputcard for a simple fcc Cu bulk system looks like

```
================ max. 500 lines, 80 chars per line, read in  ============= END|
***** Input file for TB-KKR code *****


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   R U N   &   T E S T   O P T I O N S   * * *
================================================================================
RUNOPT

+-------+-------+-------+-------+-------+-------+-------+-------
TESTOPT
ie      RMESH   clusters
fullBZ  NOSOC
+-------+-------+-------+-------+-------+-------+-------+-------
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   S Y S T E M   * * *
================================================================================
  NSPIN= 1  (1/2 paramagnetic/spin polarized calc.)
  LMAX= 3
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   B R I L L O U I N   Z O N E   M E S H   * * *
================================================================================
  BZDIVIDE=  40    40  40    (Brillouin zone mesh)
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   C O N T O U R   I N T E G R A T I O N   * * *
================================================================================
  (only 1st line is read in)
--------------------------------------------------------------------------------
  EMIN      EMAX       TEMPR      NPOL      NPT1    NPT2     NPT3
  -0.30     0.70       502.569d0    5         10       20       10     <-- for scf
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   C O N V E R G E N C E   * * *
================================================================================
  NSTEPS     IMIX      STRMIX       FCM       QBOUND    BRYMIX      ITDBRY
   1000        0         0.0200     20.0        1.D-7     0.0010      30
--------------------------------------------------------------------------------
  Various running details (scf-steps, convergence etc)
  nsteps: no. of scf-steps
  imix: 0: straight mixing, 4: broyden's 2nd, 5: anderson's
  strmix: straight-mixing parameter
  qbound: stop if potential changes less that qbound
```

```
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   L A T T I C E   &   A T O M S   * * *
================================================================================
  ALATBASIS= 6.67  1.0   1.0        lattice constants a (in a.u.), b/a, c/a
  BASISCALE= 1.0   1.0   1.0        scaling factor
  LATTICE=1
--------------------------------------------------------------------------------
  BRAVAIS                  (units of lattice constant)
  0.0   0.5   0.5
  0.5   0.0   0.5
  0.5   0.5   0.0
--------------------------------------------------------------------------------
  Parameters for the screening clusters (=: spherical cluster, !=: cylindrical)
  RCLUSTZ=  2.30d0
  RCLUSTXY= 2.30d0
--------------------------------------------------------------------------------
  Basis sites:
  NAEZ= 1    (Number of sites in the unit cell)
  CARTESIAN= F  (true: Basis in cartesian coords; false: in internal coords)
  NEMB= 0
  NEMBZ= 0
  KAOEZ= 1
  SCALING= 1.0 1.0 1.0
--------------------------------------------------------------------------------
  RBASIS
     0.00000000    0.00000000    0.00000000
SCALING=  1.0        1.0        1.0
----------------------------------------------------------------------
INTERFACE= F
NRIGHTHO=  10
NLEFTHOS=  10
<NLBASIS>=  2
<NRBASIS>=  2
LEFTBASIS    X         Y         Z     REFPOT
 -0.50000000   -0.50000000   -0.70710678  1 1
 -0.00000000   -0.00000000   -0.70710678  1 2
RIGHBASIS
  0.00000000    0.00000000   19.79898987  1 1
  0.50000000    0.50000000   19.79898987  1 2
--------------------------------------------------------------------------------
  LINIPOL= F
  XINIPOL= 30*0
  HFIELD= 0
  KHFELD= 0     (Apply ext. field in 1st iteration)
================================================================================
--------------------------------------------------------------------------------
ZPERIODL=-0.50000000   -0.50000000   -0.70710678
ZPERIODR= 0.50000000    0.50000000    0.70710678
================================================================================
  Information on the atoms:
  IRM= 349              (max no. of radial points)
  NATYP= 1              (Number of atom types, natyp > naez in case of cpa)
--------------------------------------------------------------------------------
================================================================================
  ATOMINFO
  Z    LMXC    KFG       CLS   REFPOT   NTC    FAC  IRNS   RMT     WGHT
  29.0   1    3 3 0 0    1      1       1     1.00    1  2.5542   1.0d0
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   V A R I O U S   D E T A I L S   * * *
================================================================================
  asa / full-potential:
  INS= 0        (0/1: asa/full-pot)
  KSHAPE= 0     (0/2 for asa/full-pot)
  ICST= 2      (Born steps for full-pot, 2 is ok, 3 is very good)
--------------------------------------------------------------------------------
```

```
  other approximations :
  KVREL= 0      (0/1/2: non/scalar/fully relativistic)
  KEXCOR= 2     (2/3: LDA-VWN / GGA xc-potential)
  KFORCE= 0     (forces)
  KTE= 1        (total energy)
--------------------------------------------------------------------------------
  NPAN_LOG= 15
  NPAN_EQ= 5
  NCHEB= 15
  R_LOG= 1.0
--------------------------------------------------------------------------------
  Parameters for ewald sums
  RMAX= 7.0d0     GMAX= 100.0d0     (fcc 7  65)
--------------------------------------------------------------------------------
  when set to 1 ---> writes out kkrflex_* files:
  IGREENFUN= 0
  ICC= 0
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
From here on (mostly) not needed
================================================================================
LRHOSYM= F
LCOMPLEX= F
NZ= 4
LCENTER= T
LOGINV= F
INIPOL= 0
IXIPOL= 0
CENTROFIN= 0.0 0.0 0.0
--------------------------------------------------------------------------------
MMIN= 1   MMAX=6        mmin mmax (bands)
SRINOUT= 0 0 0 0        sinn(0/1)sout(0/23)rin(0/1)rout(0/17)
--------------------------------------------------------------------------------
IRNUMX= 10
ITCCOR= 40
IPRCOR= 1
IFILE= 13
IPE= 0
KMT= 3
KWS= 2
KCOR= 1
INSREF= 0
ISHIFT= 0
IPOTOUT= 1
VCONST= 0.0d0
KPRE= 1
KVMAD= 0
KSCOEF= 0
KHYPERF= 0
KEFG= 0
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   F I L E S   * * *
================================================================================
  FILES
  4Ryshift                                  I12   (redundant)
  potential                                 I13  <- potential-file name
  madelung                                  I40   (redundant)
  shapefun                                  I19  <- shape-file name in full-pot
  scoef                                     I25   (redundant)
----------------------------------------|40-------------------------------------
  (40 chars max for file names)
================================================================================

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   * * *   D E C I M A T I O N   * * *
================================================================================
  This is used for decimation, put vacuum in the file to have vacuum
  DECIFILES
  deci1                                         left (up) host
  deci2                                         right (down) host
================================================================================
```

As can be seen the `inputcard` has a great deal of information about the system and the options of the calculation itself. When modifying the options in the `inputcard` it is very important to be careful with the formating as **the inputcard is fixed format**, that is keywords start at specific places and this order must be respected for the simulations to work properly.

Now, a proper description of each and every of the options in the `inputcard` is present in the manual for the `JM-KKR` program, so in the present only a basic explanation of the most common entries will be provided.

2.2.1 Run options

In this section the basic run options are specified, in here control parameters such as whether or not one is performing an SCF or a Density of States (DOS) caltulation. Some of the specific options which are used to be able to perform most of the `KKRSUSC` calculations are the following

- `DOS` If present the Density of States for the system will be calculated.
- `clusters` If present information about the tight-binding clusters will be printed (**Necesary for inpurity calculations**).

2.2.2 Energy contour

Now it is important to know that as the Green function is a complex quantity, the KKR method uses a complex Energy grid to realize its calculations. The details of the *contour* over which one will be working on is defined by the following entries

```
 EMIN       EMAX         TEMPR        NPOL        NPT1      NPT2       NPT3
-0.30      0.9           502.57d0      5           3         20         2
```

where `NPOL` is number of Matsubara poles. `NPT1,2,3` are number of energy points to be calculated when going up in complex axis (at `EMIN`), right in real axis (from `EMIN` to `EMAX`), and down in complex axis (at `EMAX`), respectively, whilst `TEMPR` indicates the "temperature"wchich defines the distace from the real axis. To better illustarte this one can look at Fig. 2.1. In it one can see how the energy contour is defined as a function of the different parameters specified in the `inputcard`.

Tip: Check your electrons To ensure a physically correct calculation one must check that the energy contour defined is such that all the valence electrons are actually included inside the defined area. This can be done by performing a one-shot calculation and checking that the number of electrons inside the contour is the appropiate for the studied system. This can be of great importance to describe systems that contain semicore states.

It is also critical to ensure that the Fermi energy is inside the Energy contour, a priori this is not known so the user must define a width of the contour large enough as to ensure that all the critical quantities are properly represented

2.2.3 Atomic info

2.3 `VORONOI` program

To run the `JM-KKR` program one first needs to create a series of files. One of the crucial files is the `potential` file, which as its name states will contain the information for the *starting* guess of the potential for each one of the atoms in the unit cell.

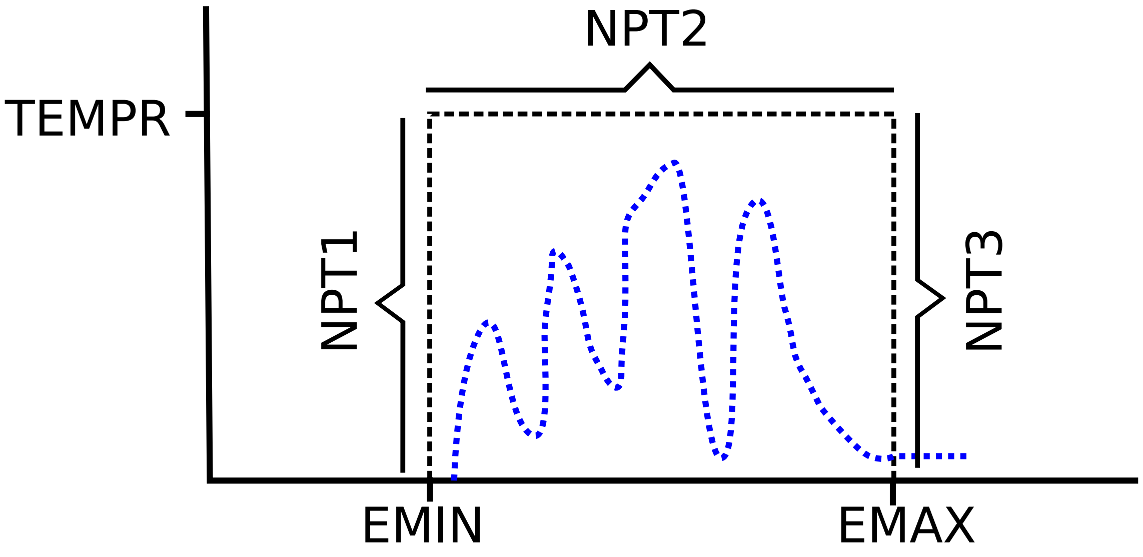


Figure 2.1: Sketch of the energy contour as defined by the parameters given in the `inputcard`.

To create this file one needs the

Before we run the KKR program, we need to create at least the potential file (possibly more files too) using the VORONOI program. This step requires of the **inputcard**, which should be basically the inputcard to be used in the KKR calculation for the new system. Then, take the time and write a proper inputcard with all relevant parameters adjusted to the new system. Copy it to the home directory of VORONOI.

We have to take special care of the 'I13' flag in the inputcard, since this points to an already existing potential file. There are two options (that I know so far):

- We dont have a previously generated potential file, i.e. we start from scratch. Then, there should be no name in the line where the 'I13' flag appears in the inputcard:

```
FILES
4Ryshift                                              I12
                                                      I13
```

- We have a potential file we want to use as a starting point for the VORONOI program. Then, we need to copy it to the home directory of VORONOI, and insert its name in the line where the 'I13' flag appears in the inputcard. (This is the case, for instance, if we have generated the **fort.3** potential file using a previous calculation.

```
FILES
4Ryshift                                              I12
fort.3                                                I13
```

)

Execute the program. The output potential is named 'output.pot', which must be copied to the folder in which we will perform the KKR calculation. Another possible output file of VORONOI is 'shapefun', used for full-potential calculations (see Sec. 2.5).

Another important thing to check at this point is the number of atoms entering the scattering problem. This is determined by the size of the TB cluster, ie the RCLUSTZ and RCLUSTXY (see Sec. 2.4.1). The important thing is that not *too many* or *too few* atoms enter: for *normal* calculations (bulk Si, Ag, Fe ...) around 60 atoms seems OK. This can be checked **at the bottom of the output file**.

One has also to check the keyword 'clusters':

```
CLSGEN_VORONOI: Atom           1  has cluster           1  with          55
 sites
CLSGEN_VORONOI: Atom           2  has cluster           1  with          55
 sites
CLSGEN_VORONOI: Atom           3  has cluster           1  with          55
 sites
CLSGEN_VORONOI: Atom           4  has cluster           1  with          55
 sites
CLSGEN_VORONOI: Atom           5  has cluster           1  with          55
...
```

If we have not entered the atomic positions or direct lattice with enough precission, VORONOI may consider atoms that should in principle be equal (environment) not equal. In the above, all atoms have been given the same cluster label.

2.4 Running KKR program

Make a new directory for running the KKR program. There, the indispensable files should be the inputcard, potential file and the executable file which will run the different KKR executables (*tbkkr_run*).

First steps that must always be done:

- Think if we need to compile the KKR program or not. In any case, **remember to set the right path-directory for the KKR executables** in *tbkkr_run*.
- Copy the potential file given by VORONOI (default name 'output.pot') to current directory. Ideally, name it 'potential', and insert this name in the line where the 'I13' flag appears in the inputcard:

```
FILES
4Ryshift                                     I12
potential                                    I13
```

2.4.1 Some parameters that are frequently changed

- Number of atoms.
  It is controlled by the parameters NAEZ and NTYPE, set them to be the number of atoms in the unit cell. Sometimes it is desirable to insert 'empty' atoms to fill the otherwise empty space between real atoms. The below describes two empty atoms introduced in bulk Si along the unit cell diagonal at (0.5 0.5 0.5) and (0.75 0.75 0.75):

```
ATOMINFO
Z     LMXC     KFG      CLS   REFPOT   NTC    FAC    IRNS   RMT     WGHT
14.0   1       2 2 0 0    1       1       1     1.00    1    2.2d0   1.0d0
14.0   1       2 2 0 0    1       1       1     1.00    1    2.2d0   1.0d0
0.0    0       0 0 0 0    1       1       1     1.00    1    2.2d0   1.0d0
0.0    0       0 0 0 0    1       1       1     1.00    1    2.2d0   1.0d0
RBASIS
        0.00000    0.00000    0.00000
        0.25000    0.25000    0.25000
        0.50000    0.50000    0.50000
        0.75000    0.75000    0.75000
```

  Thus, we set all 'zeros' in the empty atoms.
- RCLUSTZ and RCLUSTXY.

This parameters, which are in units of the **lattice constant** (not Bohr), determine the size of the TB cluster. Roughly, they determine the number of atoms that will enter in the scattering problem ( ). This should be checked in the output of VORONOI

2.4.2 SCF

Here we list the parameters that somehow are related to the SCF calculation of the potential.

- Energy contour.
  The parameters below are related to the (complex) energy integration contour:

```
EMIN      EMAX        TEMPR      NPOL      NPT1    NPT2     NPT3
-0.30     0.9          502.57d0    5         3       20        2
```

  NPOL is number of Matsubara poles. NPT1,2,3 are number of energy points to be calculated when going up in complex axis (at EMIN), right in real axis (from EMIN to EMAX), and down in complex axis (at EMAX), respectively.
- Number of iterations, potential mixing, convergence threshold (...)
  The below parameters are characteristic of an SCF calculation:

```
NSTEPS      IMIX    STRMIX      FCM       QBOUND    BRYMIX      ITDBRY
 100         4      0.050        20.0      1.D-8      0.06         40
```

  - NSTEPS, maximum number of SCF iterations, should be definitely bigger than 1.
  - QBOUND is the charge convergence threshold
  - STRMIX and BRYMIX control the percentage of the 'old' potential that is discarded (i.e the potential of the previous iteration). If we see that the charge is oscillating very much during the iterations, we may need to lower the values of these parameters. This way, the program keeps a 'larger part' of the previous potential, avoiding oscillations ( ).

2.4.3 DOS

First of all, copy the selfconsistent potential to the folder in which we will execute the KKR program for the DOS calculation. Next, we insert some changes into the inputcard that we have used for the SCF calculation.

- Insert keyword DOS:

```
RUNOPT
full inv        DOS     ...
+-------+-------+-------+-------+-------+
```

- Set appropriate energy contour integration path, i.e points only when moving along the real axis (NPT2):

```
EMIN      EMAX        TEMPR      NPOL      NPT1    NPT2     NPT3
-0.20     0.9          502.57d0    0         0      110        0
```

  Note that we can change the temperature (TEMPR) for different DOS calculations. This will change the 'smearing' used in the calculation of the delta functions for the DOS; the smaller T, the smaller smearing, thus we usually need more k-points for low T.
- Set only one cycle in the inputcard, NSTEPS=1.

2.4.4 Magnetic calculation

- Compile the KKR program with parameter KSP = 1 in file inc.p.

- KKR calculation. Start from a paramagnetic inputcard. Then, the following parameters have to be changed:

	KHFELD*	HFIELD	LINIPOL	INIPOL	NSPIN
Non-magnetic	0	0.0	f	0	1
Magnetic	1	finite value	t	1	2

(* Note how the variable 'KHFELD' is not written the same way as in the users guide 'KFIELD') This will insert a magnetic field **only at the first SCF iteration**. Then, if the starting potential was the minimum of the nonmagnetic case, we need to apply a sufficiently strong magnetic field for the first iteration so that it perturbs the potential enough to drive it away from the nonmagnetic (local) minimum. This can be, for instance, 0.1 Ry in case of Iron. We should look into the output.2 file to check if the system develops a finite magnetic moment.

- When having more than one magnetic atom in the unit cell, it is the variable `XINIPOL` that has to be used, not `INIPOL`. The program will expect to have as many integers after `XINIPOL` as different number of atoms we have, ie `NATYP`. As an example, if `NATYP=5`, then `XINIPOL 1 0 0 1 0`, where the 1's and 0's determine wether the magnetic field will be applied (1) or not (0) to the corresponding atom (this we can choose). This has not been tested by me, Prof. Ziane did it.

2.5 Full Potential (FP)

Here we describe the main steps

- If we want to use FP method, it is desirable to first run a calculation based on the 'atomic sphere approximation' (ASA). We suppose here that this is the case, i.e. that a succesfull SCF ASA calculation has been performed in the system we want to study. Then, in the SCF ASA inputcard, insert flag 'GENPOT' in the RUNOPT section:

```
RUNOPT
full inv        GENPOT  ...
+-------+-------+-------+-------+-------+
```

Run **kkr0.exe**, it should generate a file named 'fort.3', which has the right format to be read by VORONOI.

- VORONOI.
Copy the file 'fort.3' to the home directory of the VORONOI program. At this step, we basically need the inputcard for the FP calculation. For this, we need to change several flags compared to the ASA calculation. These are:

	IRNS	KSHAPE	IRM	INS	ICST
ASA	1	0	349	0	2
FP	135	2	484	1	3

Also, insert 'fort.3' name into the inputcard in the place marked by I13:

```
----------------------------------------------------------------------
FILES
4Ryshift                                              I12
fort.3                                                I13
madelung                                              I40
shapefun                                              I19
scoef                                                 I25
```

Once these flags have been changed in the inputcard, run VORONOI using this inputcard. The program should output the potential in file 'output.pot' and the shape functions in file 'shapefun'

- Compile.
  Some of the parameters above mentioned enter also in the inc.p file, so it is very likely that we may need to compile again the program. The parameters to be changed are:

	KNOSPH	IRMD	IRNSD
ASA	0	349	1
FP	1	484	208

  Compile the program with these parameters, save the executables and the inc.p file in a separate folder.
- Run KKR program
  For running the KKR program in FP mode, we need to copy the previously generated potential (output.pot) and shapefun files from VORONOI home folder to the folder where we execute the program. Also, remember to point to the right executables in the *tbkkr_run* executable file.

2.6 Band structure calculation

- To calculate with the program given by Long (zulapi.exe), we need to use the following flags:

```
RUNOPT
full invBAND-STR        ...
+-------+-------+-------+-------+-------+
```

- The k-space path is set as

```
----------------------------------------------------------------------
 DIRECNO= 1
 DIRECDEF
  50
 0.0   0.0   0.0    0.0   0.0  0.0    < Gamma
 0.0   0.0   1.0    0.0   0.0  0.0    < H
```

  Use always DIRECNO = 1, which means that only one k-space line is calculated, from Γ to H in the above case. Otherwise, if we insert more than one line, then plotting gets a bit messy due to the output format.
- Long said that the values of RCLUSTZ and RCLUSTXY should be higher than in normal scf calculations so that we take into account $\sim$ 250 neighbors.
- Set temperature to **zero** and choose the number of energy points (in real axis) and energy window:

```
----------------------------------------------------------------------
EMIN      EMAX       TEMPR      NPOL      NPT1     NPT2     NPT3
-0.20     0.8         0.0         0         0       110        0
----------------------------------------------------------------------
```

- Add variables NSPO, NSPOH and NCL_IMP to the inputcard:

```
+-------+-------+-------+-------+-------+
  LMAX=3    NSPIN=1   NATYP=4 NSPO=1 NSPOH=1 NCL_IMP=1
```

- Compiling: as in the standard KKR program, one has to change the inc.p file of the band-structure program accordingly to the used inputcard. First, make clean and `rm *o`. Then compile. Usual variables that need to be changed:

  - NATYPD controls the number of atoms in the unit cell
  - NSPDD controls the SOC
  - INSD controls whether the calculation is ASA or FP
- **Output**: the first three numbers are the coordinates of the k points in reciprocal-basis coordinates, units of $2\pi/a$. Next the real and imaginary parts of the energy, and finally a variable in which we are so far not interested. The first line corresponds to the minimum energy, and last one the maximum energy, ie increasing order.

2.7 Slab calculation

A slab calculation is meant to model a system in which periodicity is not present along at least one spatial direction. We can think of a surface, for instance. To model this, one must first prepare the position of the atoms that conform the surface, and also of vacuum layers at both sides of the surface. An example containing 9 Ag layers and 6 vacuum layers is given below:

```
CARTESIAN= t
RBASIS
0.00000000    0.00000000    0.00000000
0.50000000    0.28867513    0.81649658
1.00000000    0.57735027    1.63299316
1.50000000    0.86602540    2.44948974
2.00000000    1.15470054    3.26598632
2.50000000    1.44337567    4.08248290
3.00000000    1.73205081    4.89897949
3.50000000    2.02072594    5.71547607
4.00000000    2.30940108    6.53197265
4.50000000    2.59807621    7.34846923
5.00000000    2.88675135    8.16496581
5.50000000    3.17542648    8.98146239
6.00000000    3.46410162    9.79795897
6.50000000    3.75277675   10.61445555
7.00000000    4.04145188   11.43095213
-----------------------------------------------------------------------
ATOMINFO
Z      LMXC    KFG      CLS    REFPOT    NTC     FAC    IRNS   RMT     WGHT
0.0     0     0 0 0 0    1       1        1     1.00    1    2.3d0   1.0d0
0.0     0     0 0 0 0    1       1        1     1.00    1    2.3d0   1.0d0
0.0     0     0 0 0 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
47.0    2     4 4 3 0    1       1        1     1.00    1    2.3d0   1.0d0
0.0     0     0 0 0 0    1       1        1     1.00    1    2.3d0   1.0d0
0.0     0     0 0 0 0    1       1        1     1.00    1    2.3d0   1.0d0
0.0     0     0 0 0 0    1       1        1     1.00    1    2.3d0   1.0d0
```

Note that vacuum layers are located in both sides of the surface. This is important since for the slab calculation connects both sides to an 'infinite' vacuum (the system is not periodic along this direction in the program). Then, we need to specify the next vacuum layer in both sides, as shown below.

2.7.1 Parameters

- INTERFACE: set to TRUE
- Quit option 'full inv' from the top of the inputcard
- Set third vector of the Bravais lattice to zero:

```
BRAVAIS
  1.00000000 0.00000000   0.00000000
  0.50000000 0.86602540   0.00000000
  0.00000000 0.00000000   0.00000000
```

- LEFTBASIS, RIGHBASIS: contain the location of the 'next' vacuum layer

```
LEFTBASIS    X         Y         Z     REFPOT
     -0.50000000  -0.28867513  -0.81649658  1  1
RIGHBASIS
      7.50000000   4.33012701  12.24744871  1  1
```

- NLEFTHOS, NRIGHTHO: number of times that the vacuum layer defined in LEFTBASIS, RIGHBASIS will be repeated on top by the program
- NLBASIS, NRBASIS: number of layers declared in LEFTBASIS, RIGHBASIS, ie 1
- ZPERIODL, ZPERIODR: the real space direction in which the layers at left/right will be repeated. This is usually given by the 2nd layer contained in RBASIS, se the example below

Example:

```
INTERFACE= T
NRIGHTHO=  12    NLBASIS=  1
NLEFTHOS=  12    NRBASIS=  1
LEFTBASIS    X         Y         Z     REFPOT
     -0.50000000  -0.28867513  -0.81649658  1  1
RIGHBASIS
      7.50000000   4.33012701  12.24744871  1  1
---------------------------------------
ZPERIODL= -0.50000000  -0.28867513  -0.81649658
ZPERIODR= 0.50000000  0.28867513  0.81649658
```

2.7.2 Things to check/tricks

Before running the full KKR program, it is good to only run kkr0 and check the output file output.0. There, look for the line

```
********** TESTING THE COUPLING MATRIX ********
```

Just below of it, the program says something about a variable called NPRINCD. Still Im not sure how to properly set this value (it has something to do with the blocks of the band diagonal matrix appearing just above this point), but from Manuels advices, one should not always trust the recommendation of the program at this point. So far, using 15 layers (atoms + vacuum) we have used option NPRINCD=3. I think that if we have had an even number of layers, then NPRINCD=2 would have been preferable. This parameter has to be changed in inc.p file, and recompile the program.

2.7.3 Non-symmetric positions, relaxed structure

It is sometimes desirable to locate an impurity in a position that does not follow the symmetry of the underlying structure. This is usually done because we will eventually want to insert an impurity in that position. We know that impurities tend to approach the surface, so that its real (relaxed) distance along the surface perpendicular direction can vary as much as 25% as compared to the non-relaxed one. This can have an impact on the electronic structure.

We have to go back to the Voronoi step for constructing the potential. There, we use the new inputcard that contains the relaxed position of the impurity (on vacuum):

```
RBASIS
0.00000000   0.00000000   0.00000000
0.50000000   0.50000000   0.70710678
0.00000000   0.00000000   1.52027957 <-- this was 1.41421356 in the relaxed case
0.50000000   0.50000000   2.12132034     now it contains a 15% relaxation with respect to
0.00000000   0.00000000   2.82842712     surface layer
...
```

Above, vacuum position just on top of the surface atom has been reduced by 15%, from 1.41421356 to 1.52027957 (the distance has been reduced with respect to 2.12132034, the position of the surface layer). Note that in this case we did not change the position of the rest of vacuum sites.

We execute VORONOI with this new inputcard, and take a look to the output. The main change is that now there are more different TB cluster types. Check this part:

```
Atom            1  has cluster           1
...
Atom            1  has cluster           2
...
Atom            6  has cluster           6
Atom            6  has cluster           6
Atom            6  has cluster           6
```

Because of our modification of the position, the neighboring sites will have different TB clusters. The ones that are very far away from our modified position will have the same cluster type. In the above example, 6 different clusters were found. This is an important number, as it has to go into the **inc.cls** file for the compilation of the JM program.

So we go now to the JM folder, and recompile, inserting the correct value in the **inc.cls** file, setting the value of the parameter NCLSD to the maximum number of different clusters outputed by VORONOI: NCLSD=6 in the case studied above.

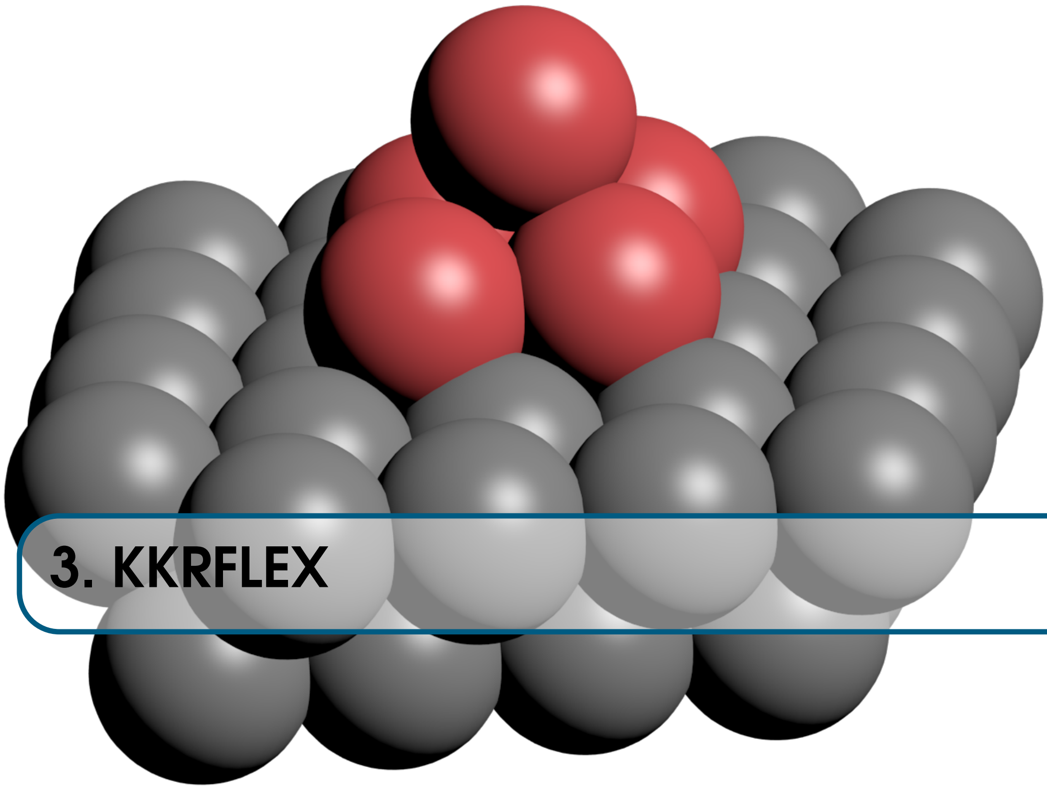
Now save and store the executables in a folder. For running the program, first copy the output potential of VORONOI, output.pot, to the folder where we will make the calculations. If we have a previously converged potential for the symmetric position of the vacuum (no relaxation), we can take advantage of it, and just copy the potential of the site that we have modified into the converged potential (maybe the neighborinig sites too). The last thing is to specify in the inputcard which type of cluster corresponds to each atom, which as we have seen is declared in the output of the VORONOI program. This is specified in the ATOMINFO part, parameter CLS:

```
                      |||
ATOMINFO              vvv
Z     LMXC     KFG     CLS    REFPOT    NTC     FAC     IRNS    RMT      WGHT
0.0     0     0 0 0 0   1       1        1     1.00     1    2.3d0    1.0d0
0.0     0     0 0 0 0   2       1        1     1.00     1    2.3d0    1.0d0
```

```
0.0     0     0 0 0 0    3      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    4      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    5      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
47.0    2     4 4 3 0    6      1       1      1.00    1   2.3d0    1.0d0
0.0     0     0 0 0 0    6      1       1      1.00    1   2.3d0    1.0d0
0.0     0     0 0 0 0    6      1       1      1.00    1   2.3d0    1.0d0
0.0     0     0 0 0 0    6      1       1      1.00    1   2.3d0    1.0d0
```

2.7.4 Version of JM code with only one executable

In the currently (04/2016) newest version of the JM code only one executable is needed. An important issue in this version is the variable `WLENGTH` in inc.p file. As far as I know, this has to do with the dimensions of the `kkrflex_*` files that are outputed for a impurity calculation (see next section). From what I understand, for "old" versions of the impurity code (the ones that currently have been connected to Manuels SOC solver) we need to set `WLENGTH=4`, while for the newest versions of the impurity code (that have to date not been connected to Manuels SOC solver) we need to set `WLENGTH=1`. If we do not take care of this, the calculation may crash at the stage of the impurity step.



3. KKRFLEX

3.1 Impurity calculation

The first step for an impurity calculation is to decide where we will put the impurity in real space. Typically, this can be in the first vacuum layer, just on top of the last surface layer. Here we follow an example of the Ag(100) surface; the starting point is the inputcard for the Ag(100) slab. Here we show the 3 vacuum layers and first Ag surface layers taken from the slab inputcard

```
ATOMINFO
Z      LMXC      KFG      CLS    REFPOT    NTC     FAC     IRNS    RMT      WGHT
0.0     0        0 0 0 0    1       1        1      1.00     1    2.3d0    1.0d0
0.0     0        0 0 0 0    1       1        1      1.00     1    2.3d0    1.0d0
0.0     0        0 0 0 0    1       1        1      1.00     1    2.3d0    1.0d0   <-- we choose this one,
47.0    2        4 4 3 0    1       1        1      1.00     1    2.3d0    1.0d0       3rd place!
47.0    2        4 4 3 0    1       1        1      1.00     1    2.3d0    1.0d0
...
```

We have indicated the location in the vacuum where we will insert our impurity, which is the one corresponding to the third position. Then, we need to extract information regarding the neighbors of this site. To do that, we have to run 1 cycle using the inputcard of the slab, with the flag **clusters**:

```
RUNOPT
clusters         ...      ...
+-------+-------+-------+-------+-------+
```

This will output a file called clusters. In there, we have information regarding the neighbors of all sites defined in our slab:

```
  18 <--number of sites
   5.4590000      ALAT
# Z       0.  0.  0. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47. 47.  0.  0.  0.
# KAOEZ    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
      55 <-- number of neighbors of each site
       1       1  #---  information regarding the neighbors of the FIRST site (as defined in inputcard)
  0.0000000000000000000E+00  0.0000000000000000000E+00  0.0000000000000000000E+00    1  0.0  0.000000000E+00
```

```
 -0.5000000000000000000E+00 -0.5000000000000000000E+00 -0.7071067811865475727E+00  -19  0.0  0.100000000E+01
  0.5000000000000000000E+00 -0.5000000000000000000E+00 -0.7071067811865475727E+00  -19  0.0  0.100000000E+01
 -0.5000000000000000000E+00  0.5000000000000000000E+00 -0.7071067811865475727E+00  -19  0.0  0.100000000E+01
^
|
| Total of 55 neighbors
|
v
      55
       2       1  #---  information regarding the neighbors of the SECOND site
  0.0000000000000000000E+00  0.0000000000000000000E+00  0.0000000000000000000E+00    2  0.0  0.000000000E+00
^
|
| Total of 55 neighbors
|
v
      55
       3       1  #---  information regarding the neighbors of the THIRD site, the one we want!
  0.0000000000000000000E+00  0.0000000000000000000E+00  0.0000000000000000000E+00    3  0.0  0.000000000E+00
 -0.5000000000000000000E+00 -0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
  0.5000000000000000000E+00 -0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
 -0.5000000000000000000E+00  0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
  0.5000000000000000000E+00  0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
  0.0000000000000000000E+00 -0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.100000000E+01
 -0.1000000000000000000E+01  0.0000000000000000000E+00  0.0000000000000000000E+00    3  0.0  0.100000000E+01
  0.1000000000000000000E+01  0.0000000000000000000E+00  0.0000000000000000000E+00    3  0.0  0.100000000E+01
  0.0000000000000000000E+00  0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.100000000E+01
 -0.5000000000000000000E+00 -0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
  0.5000000000000000000E+00 -0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
 -0.5000000000000000000E+00  0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
  0.5000000000000000000E+00  0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
  0.0000000000000000000E+00  0.0000000000000000000E+00 -0.1414213562373095145E+01    1  0.0  0.141421356E+01
 -0.1000000000000000000E+01 -0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
  0.1000000000000000000E+01 -0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
 -0.1000000000000000000E+01  0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
  0.1000000000000000000E+01  0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
  0.0000000000000000000E+00  0.0000000000000000000E+00  0.1414213562373095145E+01    5 47.0  0.141421356E+01
  0.0000000000000000000E+00 -0.1000000000000000000E+01 -0.1414213562373095145E+01    1  0.0  0.173205081E+01
 -0.1000000000000000000E+01  0.0000000000000000000E+00 -0.1414213562373095145E+01    1  0.0  0.173205081E+01
  0.1000000000000000000E+01  0.0000000000000000000E+00 -0.1414213562373095145E+01    1  0.0  0.173205081E+01
^
|
| Total of 55 neighbors
|
v
...
```

Then, from the 3rd site, we need to choose the number of neighbors that we will take into account for the impurity calculation. We can check the distance from the origin in the last column. In this example we could for example consider the first 19 ones, up to distance 0.141421356E+01. Then, we copy/paste these 19 lines from the 3rd site into a file called **scoef** (first put the number of neighbors considered):

```
19
  0.0000000000000000000E+00  0.0000000000000000000E+00  0.0000000000000000000E+00    3  0.0  0.000000000E+00
 -0.5000000000000000000E+00 -0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
  0.5000000000000000000E+00 -0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
 -0.5000000000000000000E+00  0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
  0.5000000000000000000E+00  0.5000000000000000000E+00 -0.7071067811865475727E+00    2  0.0  0.100000000E+01
  0.0000000000000000000E+00 -0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.100000000E+01
 -0.1000000000000000000E+01  0.0000000000000000000E+00  0.0000000000000000000E+00    3  0.0  0.100000000E+01
  0.1000000000000000000E+01  0.0000000000000000000E+00  0.0000000000000000000E+00    3  0.0  0.100000000E+01
  0.0000000000000000000E+00  0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.100000000E+01
 -0.5000000000000000000E+00 -0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
  0.5000000000000000000E+00 -0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
 -0.5000000000000000000E+00  0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
  0.5000000000000000000E+00  0.5000000000000000000E+00  0.7071067811865475727E+00    4 47.0  0.100000000E+01
  0.0000000000000000000E+00  0.0000000000000000000E+00 -0.1414213562373095145E+01    1  0.0  0.141421356E+01
 -0.1000000000000000000E+01 -0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
  0.1000000000000000000E+01 -0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
 -0.1000000000000000000E+01  0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
  0.1000000000000000000E+01  0.1000000000000000000E+01  0.0000000000000000000E+00    3  0.0  0.141421356E+01
  0.0000000000000000000E+00  0.0000000000000000000E+00  0.1414213562373095145E+01    5 47.0  0.141421356E+01
# put some empty lines and after the rest of the sites

  0.0000000000000000000E+00 -0.1000000000000000000E+01 -0.1414213562373095145E+01    1  0.0  0.173205081E+01
 -0.1000000000000000000E+01  0.0000000000000000000E+00 -0.1414213562373095145E+01    1  0.0  0.173205081E+01
 ...
```

We call the above the ’impurity cluster’.

Once we have created the scoef file, we need to output some files that will be needed in the impurity calculation. For this, copy the scoef file to a new folder together with the inputcard and selfconsistent potential file of the Ag(100) slab. Set these variables in the inputcard and the flag scoef:

```
RUNOPT
KKRFLEX
...
IGREENFUN= 1        ICC= 1
...
KHFELD= 0
...
FILES
4Ryshift                                      I12
potential                                     I13
madelung                                      I40
shapefun                                      I19
scoef                                         I25
```

Dont forget that the new version looks for `KHFELD`, not `KHFIELD`, scheize!! Run `tbkkr_run` (one iteration) with the recent version of the JM code by Philip (the version by Phivos seems not to output what we need). Use the executables compiled for the characteristics of the Ag100 slab. The program should output these files:

```
kkrflex_atominfo
kkrflex_green
kkrflex_hoststructure.dat
kkrflex_intercell_cmoms
kkrflex_intercell_ref
kkrflex_tmat
```

which contain information needed in the impurity run. Copy these files to another new directory where we will run the impurity program.

The final task before being able to run the impurity program is to generate the **potential file corresponding to the impurity cluster**. We take a look back to the scoef file (see above), and pay special attention to the numbers in the 4th column; this tells us the correspondence with the sites defined in the inputcard. That is, in the example above, the first position in the impurity cluster is equivalent to the third site defined in the inputcard. The next 4 lines are equivalent to the second site defined in the inputcard, and so on. In this example, we have up to 5 different sites in the impurity cluster. Then, for generating the impurity potential file we start by opening the selfconsistent potential of the slab, which contains information of all the sites included in the inputcard, 18 in this example. Then, we need to copy the potentials corresponding to the sites of the impurity cluster, given by the numbers of the 4th column. Unfortunately, **the name of the potential file must be 'potential'**, as the kkrflex program looks for a file named that way; we cannot therefore distinguish by the name if it is a impurity potential file or a 'normal' one. In this example, we need to copy the potentials corresponding to the first 5 sites. We paste them into a new potential file following the order appearing in scoef: in this example, we need to insert in first place the potential corresponding to the 3rd site. Then, insert the potential corresponding to the 2nd site **4 times**. Then, insert the potential corresponding to the 3rd site **4 times** also, and so on till we have inserted as many potentials as sites in the impurity cluster, 19 in the present example.

Use python script create-imp-pot.py, it automatically creates the impurity potential, it only needs the slab potential file and the scoef file, fancy!

Trick: for copy/pasting the correct potentials, a nice way is to write a label to the potentials we have to copy:

```
VacO POTENTIAL  HOST 1             exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.29152538
   0.00000
   3.01702   0.590108635419487   0.913965251127118
...
VacO POTENTIAL   HOST 2            exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.29152538
   0.00000
   3.01702   0.590108635419487   0.913965251127118
349
...
VacO POTENTIAL    HOST 3           exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.29152538
   0.00000
   3.01702   0.590108635419487   0.913965251127118
349
...
Ag47 POTENTIAL   HOST 4            exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.29152538
  47.00000
   3.01702   0.590108635419487   0.913965251127118
...
Ag47 POTENTIAL    HOST 5           exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.29152538
  47.00000
   3.01702   0.590108635419487   0.913965251127118
349
...
```

Then the copy/pasting is easier, and it will help to check whether the python script worked well or not.

3.1.1 Vacuum impurity

Now we are in position to perform an impurity calculation. For this, we first have to decide what atom-type will be the impurity. As a first test, we will analyze what happens if we insert an 'vacuum impurity' atom. For this, we dont need to change anything in the impurity potential file, as originally there was vacuum in the position where we have chosen to insert the impurity, and we have accordingly set a 'vacuum potential' in the first place. Clearly, vacuum is not a real impurity, and should not therefore affect the properties of the system.

We go to the directory where we have stored the output files mentioned above, the impurity potential and inputcard. Also, we need to copy the file **config.cfg**, there some variables for the calculation are defined. We then execute the KKRFLEX_source/SOURCE/**kkrflex.exe** program; this will try to find convergence selfconsistently. If we have done the things properly, the system should be near convergence starting from the first iterations, and should reach convergence soon.

Once convergence is reached, the output is written to file out_log.000.txt, and the output potential to out_potential. We can check the calculated charge density around each atom. Type

```
grep 'Atom' out_log.000.txt > impurity-charge.dat
```

and check the charge in each atom in the last sclefconsistent cycle (at the bottom):

```
   Atom    1 charge in wigner seitz sphere =    0.262375
   Atom    2 charge in wigner seitz sphere =    0.003933
   Atom    3 charge in wigner seitz sphere =    0.003933
   Atom    4 charge in wigner seitz sphere =    0.003933
   Atom    5 charge in wigner seitz sphere =    0.003933
   Atom    6 charge in wigner seitz sphere =    0.262375
   Atom    7 charge in wigner seitz sphere =    0.262375
   Atom    8 charge in wigner seitz sphere =    0.262375
   Atom    9 charge in wigner seitz sphere =    0.262375
   Atom   10 charge in wigner seitz sphere =   46.742070
   Atom   11 charge in wigner seitz sphere =   46.742070
   Atom   12 charge in wigner seitz sphere =   46.742070
   Atom   13 charge in wigner seitz sphere =   46.742070
   Atom   14 charge in wigner seitz sphere =    0.000029
   Atom   15 charge in wigner seitz sphere =    0.262375
   Atom   16 charge in wigner seitz sphere =    0.262375
   Atom   17 charge in wigner seitz sphere =    0.262375
   Atom   18 charge in wigner seitz sphere =    0.262375
   Atom   19 charge in wigner seitz sphere =   46.991713
```

This has to be compared with the charge previously calculated in the scf run for the slab:

```
grep 'Atom' output.2 > scf-charge.dat
```

that gives:

```
   Atom    1 charge in wigner seitz sphere =  0.000029
   Atom    2 charge in wigner seitz sphere =  0.003933
   Atom    3 charge in wigner seitz sphere =  0.262375
   Atom    4 charge in wigner seitz sphere = 46.742070
   Atom    5 charge in wigner seitz sphere = 46.991713
   Atom    6 charge in wigner seitz sphere = 46.999563
   Atom    7 charge in wigner seitz sphere = 46.999912
   Atom    8 charge in wigner seitz sphere = 47.000531
   Atom    9 charge in wigner seitz sphere = 46.999875
   Atom   10 charge in wigner seitz sphere = 46.999875
   Atom   11 charge in wigner seitz sphere = 47.000531
   Atom   12 charge in wigner seitz sphere = 46.999912
   Atom   13 charge in wigner seitz sphere = 46.999563
   Atom   14 charge in wigner seitz sphere = 46.991713
   Atom   15 charge in wigner seitz sphere = 46.742070
   Atom   16 charge in wigner seitz sphere =  0.262375
   Atom   17 charge in wigner seitz sphere =  0.003933
   Atom   18 charge in wigner seitz sphere =  0.000029
```

We can see that the first charge in impurity-charge.dat coincides with the third one of scf-charge.dat. In fact we can find any of the 5 different charge values present in impurity-charge.dat also in file scf-charge.dat, consistently with how we have created the impurity cluster.

3.1.2 Real impurity

Now we analyze how to introduce a 'real' impurity, that is, a real atom in top of the surface. We take as an example a Fe atom (nonmagnetic case). First, we need to accordingly change the impurity potential file, ie insert a Fe atom in place of the vacuum in the third site. To do so, we copy the slab input card to VORONOI, and in place of the vacuum, we insert the Iron atom:

```
----------------------------------------------------------------------
ATOMINFO
Z      LMXC     KFG      CLS    REFPOT   NTC     FAC     IRNS    RMT      WGHT
0.0     0     0 0 0 0     1       1       1      1.00     1    2.3d0    1.0d0
0.0     0     0 0 0 0     1       1       1      1.00     1    2.3d0    1.0d0
26.0    1     3 3 0 0     1       1       1      1.00     1    2.3d0    1.0d0
47.0    2     4 4 3 0     1       1       1      1.00     1    2.3d0    1.0d0
47.0    2     4 4 3 0     1       1       1      1.00     1    2.3d0    1.0d0
...
```

Very important: we need to use the same RMT radious as for the vacuum in order to avoid incompatibilities. VORONOI will output a potential with Fe in the third site. From this potential file, copy the part corresponding to Fe:

```
Fe26 POTENTIAL  HOST Fe             exc: von Barth,Hedin
  2.30000000  5.45900000  2.34954834
  26.00000
   3.01702   0.4092410000   0.5000000000
...
```

and replace with it the first vacuum potential of impurity-potential; namely, erase the **first** vacuum potential and insert in its place the Fe potential. This will be our starting potential for the impurity run in the 'real' case. Next, copy all the files needed by the impurity program (the kkr_ files and so on, also the config.cfg input file) to the present directory. Before running the program, we need to change the **atomic number** of the first place (now an Iron atom) in file kkrflex_atominfo:

```
0.0000000000000000        0.0000000000000000        0.0000000000000000 26.00        0        0      3
...
```

Now we can run the impurity program.

3.1.3 Magnetic impurities

Now we see how to include magnetism into the impurity calculation. First of all, we need to double the number of potentials of the nonmagnetic potential file in order to take into account the spin-up spin-down states. The easiest way is to simply copy paste every nonmagnetic potential, ie we set same (starting) potential for both spin directions. In the present example, we would have $2 \cdot 19 = 38$ potentials:

```
Fe26 POTENTIAL  HOST Fe             exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.30000000
  26.00000
   3.01702   0.5901086354   0.9139652511
349
 0.25000000D-01 0.50267689D-03
 5 1
```

```
    0  -5.12038694958D+02
    0  -5.88813480612D+01
...
```

we copy the first nonmagnetic potential (belonging to the magnetic impurity Fe) and paste it just below

```
Fe26 POTENTIAL  HOST Fe             exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.30000000
  26.00000
   3.01702   0.5901086354   0.9139652511
349
 0.25000000D-01 0.50267689D-03
 5 1
    0  -5.12038694958D+02
    0  -5.88813480612D+01
...
Vac0 POTENTIAL   HOST 2             exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.30000000
   0.00000
   3.01702   0.5901086354   0.9139652511
349
...
```

copy the second nonmanetic potential and paste it below, and so on

```
Vac0 POTENTIAL   HOST 2             exc: Vosko,Wilk,Nusair
  2.30000000  5.45900000  2.30000000
   0.00000
   3.01702   0.5901086354   0.9139652511
349
...
```

The python script of David does quickly the job, run

```
python modifypotential.py
```

in a folder containing the nonmagnetic potential file 'potential', and choose the input option 13. It will create the doubled potential 'new_potential' that we can use in a magnetic calculation.

Now we have to tell the program that we want to perform a spin-polarized magnetic calculation. This is done via the inputcard of the impurity program, namely the **config.cfg** file. There we need to change the following parameters:

- NSPIN: this variable determines if a calculation will be magnetic or not:

  ```
  NSPIN= 2
  # magnetic calculation
  # NSPIN=1 non magnetic
  # NSPIN=2 collinear magnetic calculation
  ```

  We set it to 2.
- HFIELD: this variable is used to apply a magnetic field in order to break the symmetry of the nonmagnetic calculation; otherwise the system will converge to a nonmagnetic minimum even if we perform a magnetic calculation and we know beforehand that the ground state is

magnetic. The magnetic field is introduced as

```
 #       value(Ry)    how many iterations
HFIELD=   1.E-2              1
```

The first value is the magnitude of the B field (in Rydberg), very big in the above example. The second integer number is the number of iterations in which the magnetic field is applied. One way of magnetizing our system is to apply a very strong field in the first iteration in order to strongly break the nonmagnetic symmetry. Then, the program may (should) be able to converge to a magnetic solution.
Units Note that if we want to apply a magnetic field of 1 Tesla, then we have to introduce a magnetic field of $5.7883817555 \cdot 10^{-5}/13.6d0$.

- IMIX. From experience of the KKR users, one usually needs to set **IMIX=0** (straight mixing) in order to converge to a magnetic solution, at least in the first iterations, untill the program starts to tend to a magnetic solution; then one can set **IMIX=4** (Broyden mixing), which is generally faster.

3.1.4 Impurity DOS

To calculate the DOS with the impurity program (once convergence has been reached), one has to add the flag 'ldos' to the RUNFLAG option

```
#########################################
RUNFLAG= noforce_fullgmat ldos
#########################################
```

and set only one iteration (make sure to use the converged potential). This will generate a bunch of files containing the DOS at every atom. If the calculation was magnetic, we find two files per atom (spin up/down).

Where is, however, the information regarding the integration path, number of k-points, or the temperature? All these quantities are meaningful for a DOS calculation, but do not appear in the config.cfg input file. All this sort of information is stored into the **kkr_green** and **kkr_tmat** files. The ones we have used for the scf loop of the impurity contain the setup for a scf calculation. Then, if we do a DOS impurity calculation using these files, we will get the energies along a path proper for a scf calculation, namely including points along the imaginary axis. Also the temperature is higher than in normal DOS calculations. Therefore, we would essentially not get good results.

To set the proper quantities for a DOS calculation, we need to go back to the step of generating the kkr_ files with the kkr program. That is, copy the scf potential (of the slab), the scoef file and the inputcard we used to generate the kkr_ files to a new folder (this has flag KKRFLEX in RUNOPT). Open the inputcard and change the variables pertinent for a DOS calculation: energy integration contour, temperature and k-points. Run one iteration with the program by Phillip, which will output new kkr_ files. Copy the **kkr_green** and **kkr_tmat** files to the folder in which we will perform the DOS impurity calculation. There we have to copy also all the files previously calculated in the scf run for the impurity, **except** the old files **kkr_green** and **kkr_tmat** corresponding to the scf calculation.

3.1.5 More than one impurity

Inserting more than one impurity is essentially not different to just one impurity. In particular, the DFT calculation using the JM will be the same, since we do not define any impurity there. The only difference is when building the impurity cluster. Now, instead of inserting only one real atom in one of the vacuum positions, we insert more than one. Then, usually we would like to have similar neighboring clusters around each of the impurities. Note that some of the nearest neighbors of these impurities are the same. If we would just copy/paste the neighoring positions of each of the impurities from the clusters file, we would be counting some of the positions twice, and this will crash the program. Instead, we need to use a little program by Manuel.

Needed files are:

```
inputcard  test_scoef scoef_maker
```

where `inputcard` is the inputcard of JM code, `scoef_maker` is the executable of the program and `test_scoef` is the inputfile:

```
# BRAVAIS: how many, alat, then (x,y,z)
  2  4.830  <-- lattice constant, take from JM inputcard
  1.000000  0.000000  0.000000  <-- a1 and a2, copy from JM inputcard
  0.500000  0.866025  0.000000
# RBASIS: how many, cartesian, then (x,y,z), then atomic number
  18  T  <-- how many atoms in inputcard, and if positions are given in cartesian or not
0.0  0.00000000  0.00000000 0.0  <-- positions of atoms, copy from JM inputcard
0.5  0.28867513  0.81649658 0.0
1.0  0.57735026  1.76948974 0.0
1.5  0.86602539  2.44948974 78.0
2.0  1.15470052  3.26598632 78.0
2.5  1.44337565  4.08248290 78.0
3.0  1.73205078  4.89897948 78.0
3.5  2.02072591  5.71547606 78.0
4.0  2.30940104  6.53197264 78.0
4.5  2.59807617  7.34846922 78.0
5.0  2.88675130  8.16496580 78.0
5.5  3.17542643  8.98146238 78.0
6.0  3.46410156  9.79795896 78.0
6.5  3.75277669 10.61445554 78.0
7.0  4.04145182 11.43095212 78.0
7.5  4.33012695 12.24744870 0.0
8.0  4.61880208 13.06394528 0.0
8.5  4.90747721 13.88044186 0.0
# RIMP: how many, max number of cluster atoms, then lattice coordinates, rcut, which rbasi
  3  10000 <-- first number: how many impurities we want, next number dont touch
  0  0  1.50  3 <-- first two numbers: coefficients of a1 and a2 pointing to the first imp
  1  0  1.50  3     second number; distance that determines the number of neighboring atom
  0  1  1.50  3     into the impurity cluster
```

3.1.6 Irregular geometries

For irregular geometries where the principal layer number is not very well defined, we need to make use of the newest verion of the JM code. Few points to consider

- Ewald sum
  Use the **newest version** which has implemented the 3d sum
  Otherwise it is very tricky to converge if we use the 2D sum: one has to increase the `GMAX` and `RMAX` parameters a lot, and the kkr0 step takes a few hours.
- Converge `NSHELD` parameter in `inc.p` file
  If we have many sites that are coupled we will need to increase by a lot the parameter `NSHELD`, to values ~ 30000

- Converge `NSHELL0` integer in `ALL_SOURCE_FILES/shellgen2k.f` file
  Important: `NSHELL0` must be larger than `NSHELD`, so we may have to modify by hand this parameter inside the code (default value is 10000)
- Memory:
  - the compilation can break down if some arrays run out of memory, we need to modify the line containing the flagsin the `makefile` file:

    ```
    FFLAGS       =   -r8 -traceback -module  $(OBJ)
    ```

    add the flag `-mcmodel=large`:

    ```
    FFLAGS       =  -mcmodel=large -r8 -traceback -module  $(OBJ)
    ```

    important: for some reason I now dont remember we need to compile in the `iff260` cluster, otherwise the `-mcmodel=large` is not recognized, something to do with the version of the compiler installed, always use the latest machine!
  - when running, arrays may also run out of memory at the allocation time. For instance, the integers `ISH` and `JSH` can run out of memory in main1b.f. To try to avoid it, run in the `iff597` cluster - it has 64 Gb in each node - and serially but taking the full node (ie ask for 16 CPU's but run serially).

old fashioned: gofrin

There is a patch provided by Manuel and Flaviano called godfrin. For that we need a OLD version of the JM code that incorporates this patch. Then, insert the keyword `godfrin` into the `RUNOPT` in the inputcard for the JM code, and run the kkr0.exe executable. This should output a file called `couplings.dat`. Then, copy this file to a separate folder and run there a little program called `block_partitioning.exe`. The output on the pipe out (dump it for instance to `out_block_part`) looks something like:

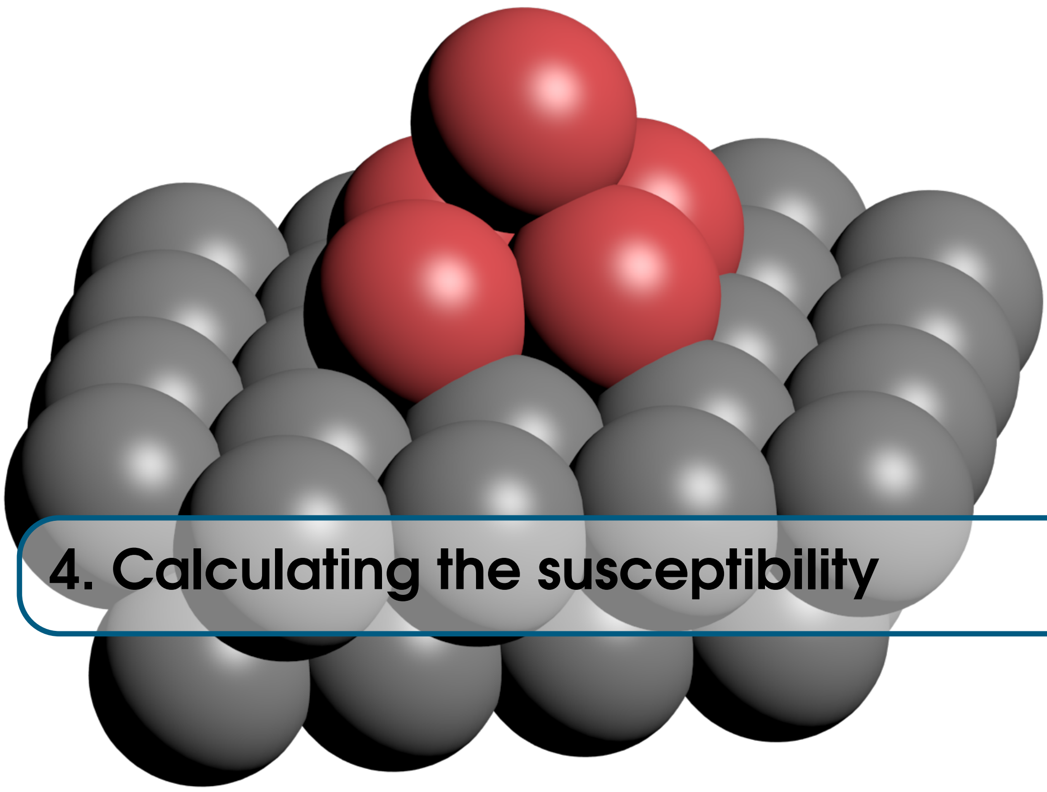
```
bandwidth: min, max, avg, dev=      27      62      48      12
minsize, maxblock=      19       6
nblocks,imax=       2      12              12
minsum2, nblocks, blocksizes=    8357       2      31      86
nblocks,imax=       3      12             144
minsum2, nblocks, blocksizes=    5049       3      30      30      57
nblocks,imax=       4      12            1728
minsum2, nblocks, blocksizes=    3429       4      27      30      30      30
nblocks,imax=       5      12           20736
minsum2, nblocks, blocksizes=    3429       4      27      30      30      30
nblocks,imax=       6       3             243
minsum2, nblocks, blocksizes=    3429       4      27      30      30      30
```

The interesting part is the last row. In this particular case it tells us that he finds 4 blocks (2nd number in last row), of sizes 27, 30, 30 and 30. So what we have to do is to copy these last numbers into a file called `godfrin.dat`, that looks like this in this example:

```
# na, nb, ldiag, lper, lpardiso; then bdims(1:nb)
 117  4  T  F  F
 27      30      30      30
```

That is, in the first line we have inserted the number of atomic sites defined in the inputcard (117, this you know from JM inputcard), and the number of blocks (4), while in the second line we have inserted the size of the blocks also given in the last line of `out_block_part`.

Finally, copy the `godfrin.dat` file to a folder and execute there the JM code with the patch, this should be able to run scf runs even in irregular geometries. IMPORTANT: copy the `godfrin.dat` to the SYSTEM folders, ie modify `tbkkr_run` file.



4. Calculating the susceptibility

4.1 KKRSUSC

4.1.1 Basic information

The KKRSUSC program is inserted **into the KKRFLEX program**, meaning that it doesnt have a own executable, the executable keeps being **kkrflex.exe**. This program currently does **not** compute the susceptibility, but it outputs the necessary files to do so by a subsequent postprocessing program. There are 2 types of files that come into play:

- Files that already exist in the original KKRFLEX program.
  These are `calctmat.f90`, `energyloop.F90`, `calctmatfull.f90` and `kkrflex.F90`. These files have been modified from the original version in order to include the stuff of the susceptibility. We have inserted the parts corresponding to the KKRSUSC program into the files corresponding to the latest version of the KKRFLEX program (before it was implemented into an older version). One should check compatibility, make tests etc.
- Files that do not exist in the original KKRFLEX program.
  These are `kkrsusc_prepare.f90` `projection.F90` and `type_inpsusc.f90`. The subroutines contained in these files are called by the files in the previous point, but these ones do not alter or modify any of the arrays used by KKRFLEX, they only compute and output new quantities, to be further processed. In order to include these files into the structure, the **makefile** has to include them when compiling together with the rest of the KKRFLEX program. These files must be copied into the SOURCE folder.

4.1.2 Runflag in impurity inputcard

To run the KKRSUSC program, first of all we need to insert the **runflag** `kkrsusc` into the **config.cfg** inputcard of the impurity program:

```
########################################
RUNFLAG= noforce_fullgmat  kkrsusc
########################################
```

This will tell the **kkrflex.exe** program that the user wants to make use of the KKRSUSC options.

4.1.3 Proper energy mesh

Before going on with inputcards and executables, it is worth to note that for calculating the susceptibility, it will be desirable to calculate the Green function and T-matrix in energy points that are different from those used in the scf calculation (of both the JM and KKRFLEX program). The energies we would like to consider are difficult to set up in the JM inputcard. The following steps have to be performed to set up the energies we want:

- Create the energy mesh file emesh.dat.
  For this purpose, there is a little fortran code named emesh.f90 that does the job. It is located in a separate folder, usually called `create_emeshdat`, look for it on the codes folder. Now we need to prepare a simple input card (lets call it inputfile):

```
 5       # npanels
-0.4    0.1      1.3    0.1     11      0       # e1r e1i e2r e2i npts imesh (1 for cl
-0.4    0.075    1.3    0.075   21      0       # e1r e1i e2r e2i npts imesh (1 for cl
-0.4    0.05     1.3    0.05    31      0       # e1r e1i e2r e2i npts imesh (1 for cl
-0.4    0.025    1.3    0.025   41      0       # e1r e1i e2r e2i npts imesh (1 for cl
-0.4    0.010    1.3    0.010   51      0       # e1r e1i e2r e2i npts imesh (1 for cl
```

  The above defines 5 lines that are parallel to the real axis, that range from -0.4 to 1.3 Ry, at imaginary energies 0.1, 0.075, 0.05, 0.025 and 0.010, using 11, 21, 31, 41 and 51 energy points respectively. The program is executed as

```
emesh < inputfile
```

  This will generate the important file **emesh.dat**, which contains all the energies in the proper format.
- (Re)Calculate the Green function and T matrix using the JM code.
  Now we have to perform a new calculation with the JM code to calculate all the stuff in the energy points contained in emesh.dat. First, copy the just created emesh.dat file to the directory where calculations will be performed. Copy also the **inputcard and converged potential** files that were used to compute the slab calculation using the JM code. Finally, copy also the **scoef** file with the appropriate information regarding the impurity cluster. Next, we need to tell the JM code that we want to use the file emesh.dat for the energies (not the path defined in the inputcard). For this, in the fortran file `SRC_COM_QDOS/main0.f` of the JM code there is a few extra lines (search for `!susc`). Then, when we add the runflag KKRSUSC to **the inputcard of the JM code** (inputcard):

```
***** Input file for TB-KKR code *****
      ***Running options***
RUNOPT
KKRFLEX KKRSUSC ...     ...
+-------+-------+-------+-------+-------+
```

  The program will look for a emesh.dat file in the directory. Note that in order to be able to use the usual `tbkkr_run` script, we need to introduce a extra line so that it copies the emesh.dat file from the directory where `tbkkr_run` is executed to the `SYSTEM` folder. This should be:

```
#==========================================================================
cp  $here/$input $local/$work_name/inputcard
#  JULEN
```

```
cp  $here/emesh.dat $local/$work_name/emesh.dat  # <-- extra line!
cp  $here/scoef $local/$work_name/scoef
```

Then, qsub the script. If everything goes well, the program will run a single scf step and ouput all the quantities calculated in the new energy mesh. Then, we need to copy the kkrflex_green and kkrflex_tmat files to the directory where we will perform the KKRSUSC calculations.
Note 1: the emesh.dat file is only needed at this step, and it is only the JM program who needs it. Once the kkrflex_green and kkrflex_tmat files have been calculated, they already contain the information on the new energy mesh. This means that when we will execute the SELFE program, even though we want to use the energy mesh defined in emesh.dat, we dont actually need this file, since its information is already in the two files mentioned above; the program will then automatically detect that the energy mesh in these files does not coincide with the energy mesh of the scf run (which in this case must be given in a separate file, see below), and proceed consequently, without the need of emesh.dat.
Note 2: sometimes putting too many panels into emesh.dat can give error messages in output.0 like:

```
     Dimension ERROR: Please increase MAXMSHD to 12
                      in the programs < main0 > and < main1b >
```

The source of this error is on how the k-mesh is re-structured by the program for different imaginary parts of the energy. It can be found in SRC_COM_QDOS/bzkmesh.f. To avoid this, use fix mesh flag in the TESTOPT part of the JM inputcard:

```
TESTOPT
ie      RMESH   clusters
verb0   fix mesh  !<-- this is the flag!
```

This will disable the re-structuring routine, and we should not get the problem anymore.

4.1.4 Proper inputcard, inpsusc.dat

Now we see what is the specific inputcard that the KKRSUSC program needs, which we name **inpsusc.dat**. It should look like something like this:

```
Data for dynamical susceptibility calculation
ne, na, nbmax, nlmax, sra=
30    1     1     2     0
ngroup, igroup(1:ngroup)=
  1
  1  3  6  3
Nb adatom
  1
nspin, iwsusc(0:lmax), ewsusc(1:nbmax,0:lmax)
2     1     0     1     0
  1     0.5901    0.010
  1     0.5901    0.010
```

The variables are as follows:

- ne: number of energy points in which the Green function was calculated by the JM code, ie the energy points present in the file inputcard. In case we have made use of the emesh.dat (see below), then the number of energy points should match with the ones in this file.

- na: number of atoms for which we compute the susceptibility and projected basis set. In the case of an impurity, if we are only interested in what happens at the impurity itself, then we would choose na=1.
- nbmax: Maximum number of wavefunctions that will be used to project the basis in any of the amgular momentum channels. That is, we can choose for instance to project the 's' channel into 1 wavefunction, but we may want to use 2 wavefunctions for the 'p' channel (for whatever reason). Then nbmax would have to be equal to 2.
- nlmax: channel with maximum angular momentum that we consider. If we consider only 's' channel, then this is equal to 0. If we consider only 'p' channel, then this is equal to 1. If we consider 's' **and** 'p' channel, then this is also equal to 1 (the disctintion between which channels to consider is done below, this only sets the maximum).
- sra=0
- ngroup: number of groups in which we will divide the **atoms** we have considered. $ngroup > 1$ could only be in the case that $na > 1$. The numbers below `ngroup` set the number of atoms in each group. In the example above, there is more than one number in the line below `ngroup` even though na=ngroup=1; it doesnt matter, the program will only read the first number as we have said that there is only one group, and ignore the rest.

Numbers inside the groups:

-
  ```
  Nb adatom
    1
  ```

  This is very important, as here we specify which atom we are interested in. Simplest example, na=ngroup=1. Even now, the program needs to know which atom among all the atoms present in the **impurity cluster** we are interested in. The label of each atom is as in the `potential` file or in the `kkrflex_atominfo`, ie usually atom number 1 is the impurity (as in this example), next ones are some vacuums and so on.

-
  ```
  nspin,   iwsusc(0:lmax), ewsusc(1:nbmax,0:lmax)
  2     !(s) 1    !(p) 0   !(d) 1     0
    1     0.5901    0.010
    1     0.5901    0.010
  ```

  Put nspin = 2 always. Important thing is taht here we choose the **angular momentum channels** to be considered. This has to do with the variable lmaxd which we have defined previously. Consider the case lmaxd=2. This means that we want to include at least the 'd' channel, but what about the 's' and 'p'? We can choose to include or not all this channels by setting 1 for yes or 0 for not 0 after declaring nspin. In the example above, the 's' and 'd' channels are considered but not the 'p' channel. Then, accordingly we need to insert 2 lines below corresponding to the 2 channels considered.

  ```
    1     0.5901    0.010
    1     0.5901    0.010
  ```

  In these lines, we first insert the label that identifies the atom (in this example we only considered the first atom [the impurity], so there is no more option than setting 1 here). The second number is the energy (in Ry) at which we cant to project the basis functions; usually the Fermi energy is of interest here. The trird number is the imaginary part of the energy, leave it at 0.01.

4.1.5 Input files

These are the input files needed to run a KKRFLEXSUSC calculation.

- `kkr_*` files.
  These can be copied from the KKRFLEX folder. We have to pay attention to the files `kkrflex_green` and `kkrflex_tmat` and whether the calculation will be performed in the scf energy mesh or that in the file emesh.dat (see Sec. 4.1.3). For the former, the two files can be taken from the KKRFLEX scf calculation, but for the later we have to create these two files in the new energy mesh, see Sec. 4.1.3 for details.
- The file `config.cfg` used in the KKRFLEX calculation.
  It is very important to realize that **all the parameters** must be **equal** to those of the KKRFLEX calculation (ie magnetic field strength and so on). Also, note that we need to put **only one iteration**. If we use the scf energy mesh, the ouput file should say that the calculation is converged. Instead, if we have used a energy mesh appropriate for calculating the susceptibility, then the output should show no convergence, an a very big scf error.
- The **converged** `potential` file, ie the **output of the KKRFLEX code**, the `out_potential` file renamed as `potential`.

4.1.6 Executing the program

The program is executed simply as the original KKRFLEX program, see Sec. 3.1. **Important**: we need to use **the same config.cfg file** as in the KKRFLEX calculation (same parameters, also applied magnetic field if any), except that we only need to do **1 iteration**. Also, make sure that we use a version of the program which includes the modifications of the SUSC part mentioned in Sec. 4.1.1.

The aim of the program is basically to compute the projected wavefunctions at the desired energy points. To see how it is going, use command:

```
tail -f outsusc.dat
```

there we should see information about wavefunctions at each energy point.

4.1.7 Output of the program

- `*.wfn` files (projected basis functions)
- `susc0001.pot`, which we **MUST** rename `susc0001.scf` to be used in next steps
- `outsusc.dat` file, to be copied for subsequent calculations

4.1.8 SCF vs NSCF calculations

As said earlier, we can use different energy meshes in the program. This will depend on our goal; if our goal is to analyze susceptibilities, the we need to use an energy mesh that suits for this calculation, which usually consists of many points close to the real axis and fewer and fewer points as we go away from the real axis. This is certainly not the energy mesh one uses for a common SCF calculation. As explained before, we need to specify the mesh in a separate file. However, everytime we study a new system, its a good check to recalculate all GS properties using the projected wvf, and compare them with the ab-initio ones, to see whats the quality of the projection scheme. Then, in general it is convenient to creat two separate folders in a directory that we can name `KKRFLEX-SUSC-fiels`. The two folders are the following:

- `scf-projected-wfn`
  Here we copy all the `kkr_*` files from a normal SCF calculation that we have runned previously. Those files contain the GF and T-matrix in the SCF energy mesh. We need **not** to copy any emesh.dat file, since we calculate on the SCF mesh. Running the program with these files will generate the `*.wfn` files on the SCF mesh. Therefore, these projected wave functions are appropriate to (re)calculate GS properties, to be checked against ab-initio calculated ones. This calculation is done using the KKRSELFE program, in the next section.

- `nscf-projected-wfn`
  Here we copy all the `kkr_*` files that we have specifically calculated using the emesh.dat file, in which an energy mesh appropriate for a susceptibility calculation has been defined. **Do not copy the `kkr_*` files of a normal SCF calculation!** We also need to copy the emesh.dat file to the folder. Running the program with these files will generate the `*.wfn` files on the mesh corresponding to emesh.dat. Therefore, these projected wave functions are appropriate for calculating the susceptibility, using the KKRSELFE program, in the next section.

4.2 Using the SELFE program; calculating susceptibilities

This is an independent program, compiled independently from the KKRFLEX/KKRSUSC program, with its own executable, **kkrselfe.x**. It is intended to be used after the projected wavefunctions have been calculated using KKRSUSC. In fact, it is only when using the SELFE program where one can check if the ouptuts of the KKRSUSC program are meaningful or not:

- Ground state properties
  The SELFE program can output ground state quantities such as charge and magnetic densities (re)calculated using the projected wavefunctions. It is always a good first check to compare these values with the ones calculated by the KKRFLEX program, since it gives an estimation of the error we are making by using the approximate scheme (also it helps if we screwed at some point the calculation, as the recalculated GS quantities will be very different)
- Excited state properties such as susceptibilities
  The real aim of the SELFE program are these kind of quantities. The quality of these will also depend on the quality of the KKRFLEX calculation, so it is here also where one can check.

4.2.1 Inputcard

```
! parameters used for the projection module:
  20   21  0.0d0  0.0d0         # numd , dend , real(eshift) , aimag(eshift) --> dnum_gf dden_gf, eshift!!!
   1 8001 -0.4d0  1.1d0  1.d-6  # idos, nedos, e0dos, e1dos, eimdos --> plot_lmdos lmdos_stps, lmdos_min lmd
! global pre-definitions for files for Green function, dyn. susceptibility, and self-energy:
  F F T                                # nondiag_ij nondiag_blocks diag
! define range for plots of lmdos, dsusc, and selfe and whether they should be written or not:
  F   -0.3e+00    +1.1e+00     0020001 # lmdos_show  lmdos_min lmdos_max lmdos_stps
  T    0.00       +3.0e-03     0000501 # dsusc_show  dsusc_min dsusc_max dsusc_stps
  F   -1.0e-03    +1.0e-03     0000401 # selfe_show  selfe_min selfe_max selfe_stps
! integration parameters for calculating dsusc and selfe:
  T T -5.0e-04 100001 -1.0e-03 100001  # int1_dsusc int2_dsusc ene_mid nene_mid ene_max nene_max
  T T +5.0e-04 100001 +5.0e-03 100001  # int1_selfe int2_selfe frq_mid nfrq_mid frq_max nfrq_max
! parameters and sampling points for fit functions:
  11 12                                 # dnum_gf dden_gf
  03 04                                 # dnum_ds dden_ds
  +8.00e-01 31 +3.00e-02 31 +1.00e-03 31 # ene_lrg nene_lrg ene_med nene_med ene_sml nene_sml (centered arou
  +1.00e-02 21 +1.00e-03 31 +2.00e-04 51 # frq_lrg nfrq_lrg frq_med nfrq_med frq_sml nfrq_sml (centered arou
! else:
  1.0e-04 # tol_global (tolerance used globally)
  F       # onepanel
```

One by one:

- First line: dimensions of fitting polinomials.
  The Green function is fitted by a polynomial:

$$G_{pq}(z) = \frac{\sum_{i=0}^{\text{numd}} a_i z^i}{1 + \sum_{j=1}^{\text{dend}} b_j z^j} \tag{4.1}$$

The parameters numd and dend are the first and second values in the first line. They have to be tested, usually around 20 is fine. Next two numbers are to shift the energy at which the polynomial is evaluated, TESTING OPTION, better set them to 0 unless we know what we are doing.

- Second line:
  - 1st parameter, idos: set to 1 (0) if want (not) to calculate the DOS
  - 2nd parameter, ndos: number of energy points in DOS
  - 3-4-5th parameters: minimum, maximum, and constant imaginary energies along which the DOS will be calculated
- Third line: 3 logical entries:
  - Write non-diagonal $i \neq j$ combinations of χ (and Σ). If we dont specifically want to check them, then set it to False, this is lot of information
  - Write non-diagonal s-p, s-d, p-d blocks. Again, set to False normally.
  - Write diagonal elements s-s, p-p, d-d... This is usually the interesting part, set to True normally.
- 4-5-6th lines: output file for plotting.
  Here we decide wheter files for plotting DOS (4th line), susceptibility (5th line) and self-energy (6th line) should be outputed and how. In each file the parameters we have to enter are the same:
  - 1st parameter: logical character to set wheter the file will be outputed or not
  - 2nd and 3rd parameters: emin and emax (real part)
  - 4th parameter: number of points
- 7-8th lines: integration parameters

4.2.2 Energy mesh file of scf run: emesh.scf

Since most of the times we will be interested in using Green functions and T-matrices calculated in a energy mesh different from that used in the scf run, we have to somehow tell the SELFE program which was the original energy mesh of the scf run; the program needs it because he will perform the complex contour integration for getting the KS susceptibility, the so called I_1 integral (see [**samir**]). Note: the scf energy mesh is usually a convenient path for this integration, but it is not by all means the only possible or proper path. Then, we need to give the scf original mesh explicitly to the program, with the proper weights of each energy points. This is properly done as follows:

- Go to folder where the scf run for the JM code (slab calculation) was done
- In the input card (file inputcard) add the following flag:

  ```
  ***** Input file for TB-KKR code *****
        ***Running options***
  RUNOPT
  emesh   ...
  ```

  and run 1 iteration.
- The program should output a file called **emesh.scf**, which should be copied to the folder where the SELFE program will be executed.

4.2.3 Executing the program, files that are needed, files that are created

- **Input files**
  - First, copy the following files from the KKRSUSC folder:
    - all `*wfn` files (containing projected wavefunctions)
    - all `kkr_*` files
    - `outsusc.dat` file

    - `susc0001.pot`, and **important**, copy it to a file called `susc0001.scf`, ie command line
      `cp susc0001.pot susc0001.scf`
  - `emesh.scf` file previously calculated by the JM code
  - inputcard `input.selfe`
  - `config.cfg` file for reading the magnetic field (new version)
  - `lebedev_ascii.gga` file, just copy it there, always same file
  - `excorr.krnl`: exchange correlation kernel. This is an input/output file, as the program will output it if he finds no such file, but will not output if he finds. Right now, the program is built such that the xc kernel that has to be used should be calculated when **no magnetic field is applied**. This means that even though we are performing a calculation where a finite magnetic field was applied selfconsistently, we need to use the `excorr.krnl` outputed when there was no magnetic field. (see paper by Samir)
- **Output files**:
  - charge and magnetic densities, ouputed in the pipeout. Grep for `GS quantities for ia`, they are there below.
  - Susceptibilities (KS and interacting): `dsusc_ia001ja001.dat`
  - (Exchange correlation kernel: `excorr.krnl`)

4.2.4 SCF vs NSCF calculations

In Sec. 4.1.8 we saw the difference between scf and other type of energy mesh-es. With the KKRSELFE program, we can calculate different type of properties depending on the mesh.

- `scf-projected-wfn`
  Here we copy all the `kkr_*` files from a normal SCF calculation that we have runned previously. Those files contain the GF and T-matrix in the SCF energy mesh. We need **not** to copy any emesh.dat file, since we calculate on the SCF mesh. Running the program with these files will generate the `*.wfn` files on the SCF mesh. Therefore, these projected wave functions are appropriate to (re)calculate GS properties, to be checked against ab-initio calculated ones. This calculation is done using the KKRSELFE program, in the next section.
- `nscf-projected-wfn`
  Here we copy all the `kkr_*` files that we have specifically calculated using the emesh.dat file, in which an energy mesh appropriate for a susceptibility calculation has been defined. **Do not copy the `kkr_*` files of a normal SCF calculation!** We also need to copy the emesh.dat file to the folder. Running the program with these files will generate the `*.wfn` files on the mesh corresponding to emesh.dat. Therefore, these projected wave functions are appropriate for calculating the susceptibility, using the KKRSELFE program, in the next section.

4.3 Calculating self-energies and renormalized DOS

There are two main steps.

- Projection step
  We need to copy:
  - converged potential from impurity code `out_potential`, rename to `potential`
  - `config.cfg` file
  - `emesh.scf` file, it must contain a energy contour proper for a scf calculation. This is very important as **the KS susceptibility is calculated along this contour**. Note that the Green function and T-matrices that we will provide in this step are **not** calculated in the scf contour, see below.
  - `kkrflex*` files, in particular, `kkrflex_green` and `kkrflex_tmat` have to be calcu-

lated in a panel of energies where there are a lot of points close to real axis (energy typically must range from the valence band up to well above the Fermi level), and less and less points as we go into the imaginary axis (note this pannel is different from what Manuels program uses for susceptibility calculation.) This is a typical example of the panel structure:

```
 5       # npanels
-0.4    0.1      1.3    0.1     11      0       # e1r e1i e2r e2i npts imesh (1 f
-0.4    0.075    1.3    0.075   21      0       # e1r e1i e2r e2i npts imesh (1 f
-0.4    0.05     1.3    0.05    31      0       # e1r e1i e2r e2i npts imesh (1 f
-0.4    0.025    1.3    0.025   41      0       # e1r e1i e2r e2i npts imesh (1 f
-0.4    0.010    1.3    0.010   51      0       # e1r e1i e2r e2i npts imesh (1 f
```

The program will then **fit** the Green function using those energy points and **interpolate** it to the energy points of the scf contour so that it can integrate and calculate the susceptibility.

  – `inpsusc.dat`
  – `input.selfe`

converged potential from impurity code,

- jj

4.3.1 Applying constant magnetic field

To activate/deactivate using the projected or fitted green functions, go to projection.F90 and check for JULEN

4.3.2 Code

The tree: `write_dsusc` calls `calc_kssusc`, has subroutine `calc_kssusc_int1`

Variables in `projection.F90` (there are a lot):

```
- nlmsb: sets the total size of susceptibilities: angular x spin x basis size,
         its usually of order 10 (checked from output for Rh adatom)

- nlmsba: size of GF blocks, dimension: nasusc

- nasusc: number of atoms for susc calc
```

4.4 Impurity solver with spin-orbit interaction (Manuels program)

4.4.1 Notes on the code

Kernels

Choosing which kernel to include are handled by logical variables in the inputcard, `lkha` (Hartree kernel), `lkxc` (xc kernel). The Hartree Kernel is build in a independent subroutine `build_khartree.F90`.

KS susceptibility

Kohn-Sham susceptibility is built using the static one, and a Taylor expansion of order 2.

4.4.2 Running the code, inputcard

Files needed to run a scf run with the program:

```
config.cfg         kkrflex_hoststructure.dat  kkrflex_tmat    potential
kkrflex_atominfo   kkrflex_intercell_cmoms    send-kkrsolver.exe
kkrflex_green      kkrflex_intercell_ref      newinpsusc.dat
```

For susceptibility calculation we need to include a file called `meshpanels.dat` too. This file informs about the energy mesh, which in a susceptibility calculation is usually different from the scf contour mesh. The file must be of the form:

```
 5       50  # npanels, netot
-0.4     0.0      -0.4     0.5      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
-0.4     0.5      0.59010 0.5      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
 0.59010 0.5      0.59010 0.2      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
 0.59010 0.2      0.59010 0.02     10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
 0.59010 0.02     0.59010 0.0      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
```

Note that this is almost the same as the `ein_chebypan` file used to create the energy mesh:

```
 5     # npanels
-0.4     0.0      -0.4     0.5      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
-0.4     0.5      0.59010 0.5      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
 0.59010 0.5      0.59010 0.2      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
 0.59010 0.2      0.59010 0.02     10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
 0.59010 0.02     0.59010 0.0      10       1        # e1r e1i e2r e2i npts imesh (1 for cheby)
```

The only difference is that in `meshpanels.dat` we need to explictly write the number of energy points used (50 in this example).

Input parameters, newinpsusc.dat

The meaning of most of the input parameters are explained in the template `newinpsusc.dat`. Here some that are not:

- Susceptibility options
  - `ldynsusc` Logical variable. It determines wether only the static susceptibility (at $\omega = 0$) or the dynamic susceptibility (finite ω) will be printed
  - `lenhanced`. Logical variable. If True, it calculates and prints the enhanced susceptibility, if False it does not calculate the enhanced one, and in principle it does not print neither the KS one, ie it does not create a `evals.dat` file.
  - `lkxc`. Logical variable. If true, it uses the Kernel, if False not. It can be useful if we want to print the KS susceptibility, we can just set this term to False and look in file `evals.dat`.

Anisotropy energy

Variables for rotating the spin direction involve

```
ispinrot= 0  urot= 0.0000  0.0000  1.0000  dirmix= 0.0000
```

We need to put `ispinrot= 0` (not 1) so that the program rotates the spin direction. `urot` sets the direction to which it is rotated.

Relax the direction of the magnetic moment

Start from a converged potential with SOC, spins not rotated (pointing along z axis). The first job is to create the file `magdir.dat` where the directions where the spins should point for each atom of the cluster are written. For this, we run **just one iteration** with the following flags turned on in the `newinpsusc.dat` file:

```
lrot= T ljij= T lsoc= T
...
 lgrefsph= T
 ispinrot= 1  urot= 0.0000  0.0000  1.0000  dirmix2= 0.5000 dirmix3= 0.4000
```

This will output the `magdir.dat` file. In the following, we need to converge the direction of the spin moments; for this, increase the number of iterations, and copy `magdir.dat` to the folder where the program will be executed. From now on, the program will output a new `magdir.dat` each time it finishes a new iteration (kind of like with the potential file). One has to take a look to the directions of the spins while the convergence, since they can start pointing in directions that break the symmetry of the lattice due to numerical errors; in such case, one has to stop and restart from a previous iteration.

(Why do we need the J_{ij} tensor (`ljij= T`), if that is a thing entering the Heisenberg model, while we are using ab-initio approach? The way in which the magnetic moment is relaxed in the program actually needs these coefficients, see Manuels notes.)

DOS calculation

We do not have to feed any file which contains explictly the energy points, the program will just calculate the DOS at the energy point at which the Grren function has been calculated. Therefore, we only need to feed an appropriate Green function calculated along a line parallel to the real axis.

Different atom groups

Out of the impurity cluster we have to decide how many atoms we consider, and set them into groups. For instance, we can include the impurity into group 1 and the rest of the cluster to group 2. This would be the corresponding parameter setup in `newinpsusc.dat` for a cluster of total 43 sites:

```
 lwarn= T
 ikkr= 2  ldos= F  lsoc= F  lldau= F  lbfield= F  lsusc= F
 lrot= F  lfit= F  ljij= F  lsemodel= F  lhdio= F

# Number of energies, atoms, max basis size, lmax, max radial points, max spin
 nesusc= 50  nasusc= 43 nbmax= 8  nlmax= 3  isra= 0  nrmax= 353 nsmax= 2

# Groups of atoms to be treated in the same way
 ngroup= 2
 igroup= 1 42

# Which atoms belong to which groups
 ig=  1  ia=   1
 ig=  2  ia=   2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

Note that nesusc is a very important variable, it has to be equal to the energy points of the Green function!! Note that this applies not only when we set lsusc true is not only the energy points for susceptibility calculations, it is energy points in ger

4.4.3 Output files

- `evals.dat`
  Here we find the diagonal terms of the enhanced susceptibility. The piece of code that prints it is (`susceptibility/dyn_susc_expansion.f90`):

  ```
      write(iofile,'(1000es16.8)') zw(iw), evals(:), 1.d0/evals(:)
  ```

  the dimension of `evals()` is `2*nasusc2`, ie twice the number of atoms chosen for susceptibility calculation. So for a single atom, `evals` is a 2 entry object containing the eigenvalues of $\chi_{\pm}$ and $\chi_{\mp}$ (remember these are complex numbers so this gives 4 numbers). Then the inverse `1.d0/evals(:)` takes as many numbers.

Getting exchange parameters

Dzyaloshinskii-Moriya

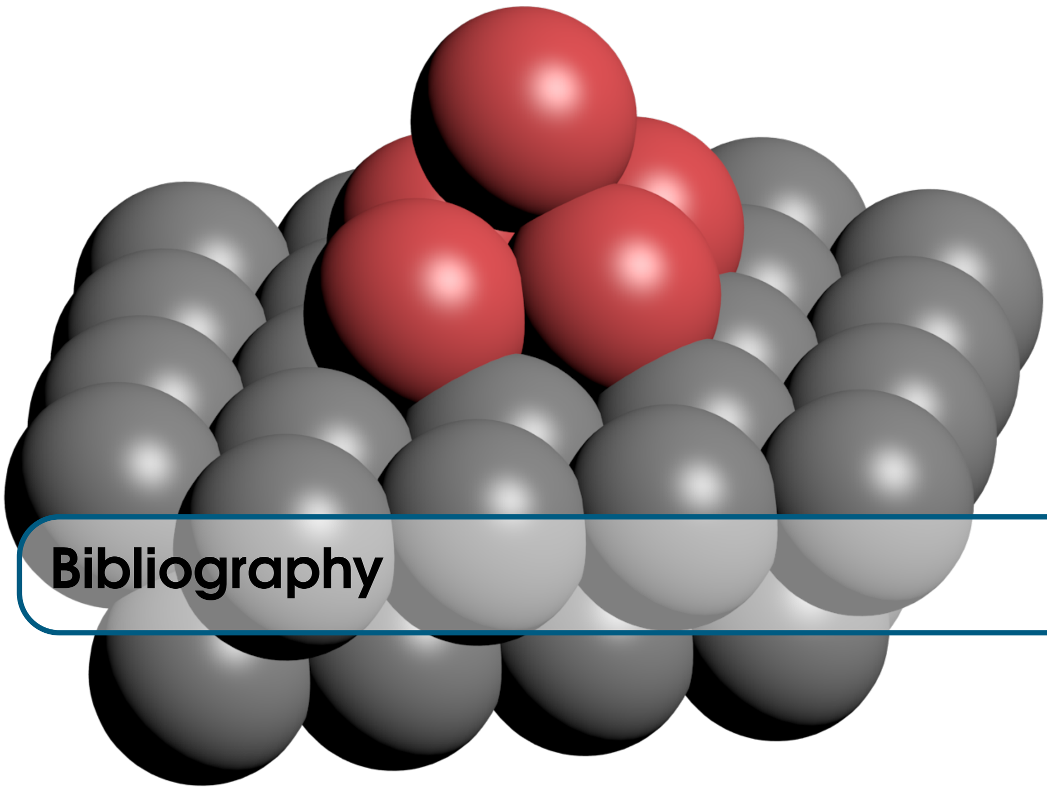
4.5 Manuels program to build geometry

Try with version 3, builder_3. Then

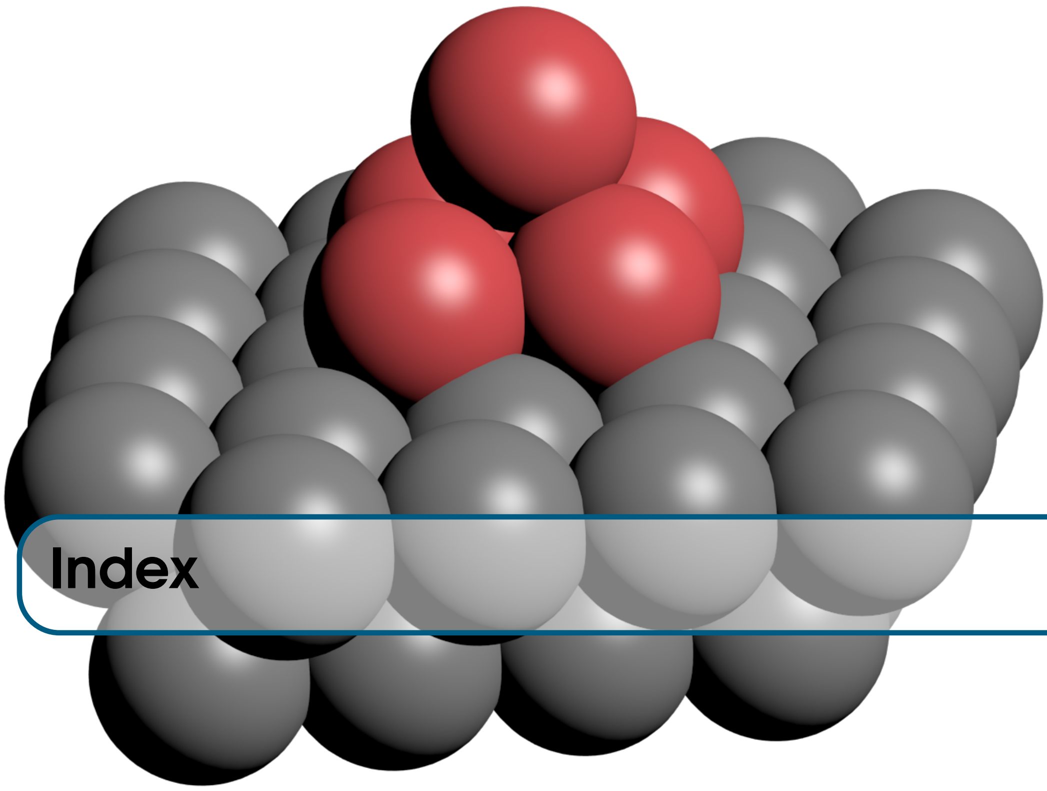
```
Set smt = sws? 1 for yes, 0 for no:
1
Number of principal layers (NPRC):  #-> number of layers we want to consider!
15
Number of extra PLs for screening (NEXTRA):
0
Number of layers in each PL on interface (1:NPRC):
1
1
... (type 15 times 1)
Number of layers in left PL (NINPRCL):
1
Number of layers in right PL (NINPRCR):
1
Layer relaxation? 1 for yes, 0 for no
0

Stacking fault?   1 for yes, 0 for no
0
```

The output is in file input_geo.in_nrsp



Bibliography



Index