ecalj — Get statrted

Takao Kotani

September 1, 2014

Contents

1 Introduction

The `ecalj` is an first-principles electronic structure calculations package in the DFT and GW. It has some unique features. Especially, we can perform the quasiparticle self-consistent GW (QSGW) calculations based on the PMT method (=the Linearized APW+MTO method) [Kotani(2014), Kotani and van Schilfgaarde(2007)]. We name this the PMT-QSGW. Introduction to PMT-QSGW is given in Sec.1.1.

The ecalj web site is at `https://github.com/tkotani/ecalj`
See README shown at the page first. Free to download `ecalj` package from it, and use it. The QSGW code is version controlled by git. Install and minimum tests are easy; even in a note PC (e.g, we can use gfortran in Ubuntu 12.04 on Thinkpad T420s. You can reproduce things in this manual in a day or so). For productive calculations, it may be needed to use a node with multi cores. Current implementation for parallelization by MPI is limited (not so much especially for the dielectric funciton part yet). So, probably, it is not so efficient to use multiple nodes simultaneously now. We expect you to clarify acknowledgment to `ecalj` in your publications.

The `ecalj` is related to a FP-LMTO package lmv7 seen at
`http://titus.phy.qub.ac.uk/packages/LMTO/fp.html`. The lmv7 and ecalj are branched off at year 2009. After branched, contributions are due to T.Kotani and Hiori Kino (NIMS) until now. We added new features: simple install and test; all codes are in f90 (no C compilar); new methods, especially PMT-QSGW; MPI parallelization for QSGW; simple usage with automatic setting of default files by python ver2; a small tool to convert VASP POSCASR to ecalj, and so on. PMT-QSGW shows more stable convergence than the previous version of FP-LMTO-GW [1].

This document shows minimum about how to plot energy band, density of states, and partial dos after we perform the LDA/QSGW calculations. It is possible to calculate quantities such as dielectric funcitons, magnetic fluctuations, after the self-consistent calculaitons.

If you have something, let takaokotani(at)gmail.com know it.

1.1 Features of the ecalj package.

PMT: A central part in an electronic structure packages is one-body problem solver. It means how to calculate eigenvalues/eigenfunctions for a given one-body potential. Inversely, we have to generate new one-body potential for given eigenfunctions/eigenvalues based on the density functional theory (DFT) in the LDA or GGA (In the followings, LDA means both of LDA and GGA). Then we can make the electron density self-consistent by iterations until converged, and obtain total energy of ground states. Then we can calculate atomic forces by perturbation. Based on such an one-body problem solver, we can implement

[1] In cases to treat magnetic systems which have intrinsic magnetic fluctuations, we may need to be careful about initial condition or mixing procedure to get convergence. In cases, we need to start from LDA+U results as initial condition from which we start QSGW. Let me know about such trouble.

kinds of methods; e.g, dielectric function, magnetic susceptibility, transport and so on. Furthermore, we can implement higher-level approximations such as the QSGW method explained below. An one-body problem solver (in linear methods) are characterized by
(i) linear combinations of what basis set to represent eigenfunctions;
(ii) how to represent electron density and one-body potential.
In `ecalj`, we use the PMT method [Kotani and van Schilfgaarde(2010), Kotani and Kino(2013)] as the one-body problem solver. The PMT method is a new all-electron full potential method. It uses not only the augmented plane waves (APW) but also the muffin-tin orbitals (MTO) together, in addition to the local orbital (lo), to represent the eigenfunctions (no other methods use two kinds of augmented waves together). Thus eigenfunctions are expanedd in the linear combinations of the APWs, MTOs, and the lo's. For electron density and the one-body potential, they are given by the three components representation. That is, the electron density (one-body potential) is divided into three components,
"smooth part + onsite muffin-tin (MT) part − counter part".
Here the counter part is in order to remove smooth part within MTs. This formalism (Soler-Williams formalism) is also used in the projected augmented wave (PAW) method [Soler and Williams(1989), Soler and Williams(1990)].

We now usually use highly localized MTOs together with APWs of low energy cutoff ($3 \sim 4$ Ry).[2] I think this is promising not only for efficient DFT/QSGW scheme, but also for kinds of applications in future. The MTOs can play a role of the Wannier functions.

QSGW: In `ecalj`, we can perform the GW calculation. The usual GW approximation is so-called "one-shot GW" starting from LDA [Kotani and van Schilfgaarde(2002), Friedrich *et al.*(2010)Friedrich, Blügel, and Schindlmayr]. It is usually only calculating differences between the quasiparticle energies (QPEs) and the LDA eigenvalues by a perturbation (only diagonal part of self energy for the LDA eigenfunctions). Its ability is limited; it can fail when its starting point (eigenfunctions and eigenvalues supplied by LDA) is problematic. Thus T.Kotani with collaborators developed the QSGW method. The QSGW now becomes popular and taken as a possible candidate to go beyond current limieation of such GW and LDA/GGA. In principle, results given by QSGW do not depend on LDA anymore; the LDA are only used to prepare initial condition for self-consistency iteration cycle of the QSGW calculation [3].

Usually the QPEs obtained by QSGW reproduce experiments better than LDA. For example, the band gap by GGA for GaAs is about 0.5 eV in contrast to the experimental value of 1.69 eV [4] On the other hand, the QSGW predicts about 1.8$\sim$ 1.9eV, a few tenth of eV larger than experiment (for practical use,

[2]current implementation have not yet efficiently use this locality; this must allow us to speed up one-body problem solver.

[3]Exactly speaking, we use LDA idea for efficient implementation of QSGW; thus obtained results are slightly dependent on the choice of LDA or GGA

[4]We undo electron-phonon effect (0.06eV) and spin-orbit effect (0.11eV) from the true the experimental value 1.52 eV.

we sometimes use "hybrid functional between QSGW and LDA" so as to obtain smaller band gap). Even in the case of NiO and so on, the QSGW gives reasonable results (there is a tendency to give a little larger band gaps than experiments). This is in contrast to the case of the one-shot GW applied to NiO, where we can not have good agreement with experiments because the stating points in LDA is problematic.

The `ecalj` have other functions. LDA+U, atomic forces and relaxation (in GGA/LDA), core level spectroscopy and so on. In addition, we can calculate dielectric functions and magnetic responses from QPEs and the quasiparticle eigenfunctions given by LDA/QSGW. But total energy in QSGW is still in research (shown total energies in QSGW calculations are dummy now).

The QSGW calculation requires so much computational time: roughly speaking, it takes 10 or more times expensive than usual one-shot GW (although we can reduce computational time by choosing computational conditions). Thus the size of systems which we can treat is limited to ten atom in a cell or something, say, with a node of 16 cores; computation may require a week or so to have reasonable convergence. (heavy atoms require longer computational efforts, light atoms faster; non-magnetic systems are easier. We still have much room to accelerate the method, but not have done yet so much. Minimum MPI parallelization is implemented). The computational effort is $\propto N^4$ in the most time-consuming part of QSGW.

1.2 What do we expect for QSGW?

Let us recall hybrid functionals such as B3LYP, and LDA+U. In hybrid functional methods, we use Vxc = (1-α)*LDA+α*(Fock exchange like term), where α is taken to be ~ 0.25 usually. The α can be dependent on materials; for metals α should be almost zero. For larger band gap insulator, α becomes larger. [5] Despite of success of the functional, its ability is limited. For example, it is known that a hybrid functionals fail to describe metals such as bcc Fe. On the other hand, we have LDA+U method which succeeded to describe materials including localized electrons. However, it contains kinds of ambiguity and U is chosen by hand. The important part of the hybrid functional methods and LDA+U is the non-local potential. It is missing in the DFT. As we discussed above, they give some success but not satisfactory. We somehow need to have a method to determine high-quality non-local potential (a substitution of the exchange-correlation potential). It is the QSGW method.

Note two important aspects of non-local potential (missing character in the local potential used in DFT). One is the onsite non-locality; it is also taken into account by LDA+U model. However, note that relative shift of O(2p) band with respect to the center of 3d band is not in LDA+U. The other is the off-site non-locality (mainly between nearest neighbors), which may relate to LUMO-HOMO gap. A non-local potential can behave a projector which push down only the HOMO states (valence band) to lower energy. This can be in the

[5] If you use $\alpha = 1$ (Hartree-Fock limit), the band gap of Si becomes 20eV or something.

hybrid functional but not in LDA+U.

In the QSGW, we determine such a non-local potential with the calculation of the GW method, in a self-consistent manner (we repeat GW calculations until converged). We can expect QSGW much more than hybrid methods/LDA+U. Very roughly speaking, because the QSGW automatically determine U of LDA+U, or alpha of the hybrid functionals. More accurately speaking, we determine not only G_0 but also W (the screened Coulomb interaction) self-consistently. Here W corresponds to U and alpha. Thus QSGW gives reasonable results even if it is applied to metals such as Fe. For systems with metallic screening, it gives small non-locality (results are close to those of LDA). For systems with large band gap, QSGW gives large enough non-locality (like 0.25*(Fock exchange)).

Since we now need to treat complex systems, e.g, metal on insulator, it is very essential to treat kinds of materials on a same footing.

The main purpose of QSGW is to determine an one-particle effective Hamiltonian H^0 [6], which describes the quasiparticle picture (or independent-particle picture) for the system we calculate. In other words, QSGW divide full many-body Hamiltonian H into $H = H^0 + (H - H^0)$. The screened Coulomb interaction W is determined self-consistently in the QSGW iteration cycle.

In comparison with LDA, we see differences;

- Band gap. QSGW tends to give slightly larger than experiments. It looks systematic.
- Band width. Usually, sp bands are enlarged (except very low density case such as Na). This is the case for homogeneous electron gas. As for localized bands like 3d electrons, they can be narrowed.
- Relative position of bands. e.g. O(2p) v.s. Ni(3d). More localized bands tends to get more deeper. Exchange splitting between up and down (like LDA+U) get larger. In cases such as NiO, magnetic moment become larger; closer to experimental values.
- Hybridization of 3d bands with others. QSGW tends to make eigenfunctions localized.

However, reality is complexed, and not so simple in cases.

2 Install

Look into ecalj/README. It is also shown at `https://github.com/tkotani/ecalj`. Installation of ecalj is not so difficult (especially for gfortran and ifort). I myself use Ubuntu12.04+gfortran+note PC for development of ecalj. After install procedure finished, we will have all required binaries and shell scripts in

[6] people often pronounce this H-naught

your ~/bin/ directory. (or somewhere else where you specified in Makfile and make.inc(BINDIR)). We have automatic installation checker.

3 LDA/GGA calculations and Plots

Calculations are performed by following steps. These steps are detailed in the following sub-sections.

To identify a set of files used for a material we calculate, we use an extension to files. For example, files explained below are with extensions (only lower case allowed) of materials. For example, `ctrls.cu` and `ctrl.cu`. In this case `cu` is the extension. Any extension works. Other possible examples are `ctrls.lagao3`, `ctrl.wgantest1`, and so on.

1. Write crystal structure file `ctrls.*`, which contains crystal structure. It can be by hand, or convert it from POSCAR (in vasp). There is a tool to convert between POSCAR and ctrls. (ecalj/StructureTool/README).

2. Generate `ctrl.*` from `ctrls.*` by a script `ctrlgenM1.py`. Here `ctrl.*` is the main control file which contains all required information to perform calculations. the content in `ctrls.*` in included in the `ctrls.*` as a part. If necessary, we edit the generated `ctrl.*` file before next step.

3. Check crystal structure. `lmchk` is to confirm the crystal structure (space-group symmetry and so on). `lmchk` is applied not to `ctrls.*` but to `ctrl.*`. `ctrls.*` never used in the following steps.

4. Run `lmfa` (just calculate spherical atoms (MT sites) placed in the cell). It also calculates core eigenfunctions and valence electron charge to set up initial condition. Then we run main calculation of LDA by `lmf`. It repeats iterations, and end up with converged results in LDA. Main result (electron density satisfying self-consistency) is stored in restart file `rst.*` (binary file).

5. Plot energy band, DOS, PDOS, by running scripts. It is quite easy. Since we use gnuplot to plot them, meanings of obtained data is apparently clear.

3.1 Write crystal structure file, ctrls

Let me show some samples of crystal structure files `ctrls.*`.

Cu: `~/ecalj/lm7K/TESTsamples/Cu/ctrls.cu`

```
------from here ------------------
% const da=0 alat=6.798
STRUC  ALAT={alat} DALAT={da}
       PLAT=  0.0 0.5 0.5   0.5 0.0 0.5    0.5 0.5 0.0
SITE    ATOM=Cu POS=0 0 0
------to here ------------------
```

GaAs: ecalj/lm7K/TESTsamples/GaAs/ctrl.gaas

```
------from here ------------------
#id  = GaAs
%const bohr=0.529177 a=5.65325/bohr
STRUC
     ALAT={a}
     PLAT=0 0.5 0.5  0.5 0 0.5  0.5 0.5 0
SITE
     ATOM=Ga POS=0.0 0.0 0.0
     ATOM=As POS=0.25 0.25 0.25
------to here ------------------
```

SrTiO3: ecalj/lm7K/TESTsamples/SrTiO3/ctrls.srtio3

```
------from here ------------------
%const da=0 au=0.529177
%const d0=1.95/au a0=2*d0 v=a0^3 a1=v^(1/3)
HEADER  SrTiO3 cubic
STRUC   ALAT={a1} DALAT={da}
        PLAT=1 0 0  0 1 0  0 0 1
SITE
      ATOM=Sr POS=1/2 1/2 1/2
      ATOM=Ti POS= 0   0   0
      ATOM=O  POS=1/2  0   0
      ATOM=O  POS= 0  1/2  0
      ATOM=O  POS= 0   0  1/2
------to here ------------------
```

Lines starting from '#' are neglected as comment lines. Lines starting from '`% const`' define variables and set values (in these cases, `da`, `alat`, and `bohr`, and so on). Then the variable `alat` is referred to as `{alat}`; in the cu case, `{alat}` means 6.798. Lines not start from "#" nor "%" are main content in the ctrls file. [7]

Note that we have two tags of "categories" "STRUC" and "SITE". ("HEADER" tag is also; but it is just for user's memo shown in console output). These tags should start from the first column. Thus ctrls is divided into multiple "categories". In a category, we have "tokens" such as ALAT, DLAT, PLAT. These under STRUC category. ALAT+DALAT specify unit of length in this ctrl file. These are in a.u. (= bohr radius=0.529177Å).

The unit cell is given by PLAT (as noted, ALAT+DALAT as unit). In the above example of GaAs, three primitive cell vectors specified by nine numbers after PLAT=; they give three primitive vectors; PLAT1=(0,0,0.5), PLAT2=(0.5, 0.0, 0.5), and PLAT3=(0.5, 0.5, 0). DALAT is convenient to change lattice constant; but it is fixed to be zero here; thus no effect in this example.

[7] For these variables, we can overlay values when we start programs. For example,'lmf -vdalat=0.1 si' means that `alat` is recorded in save.si file.

Note that SITE category can have multiple ATOM tokens. The number of ATOM token under SITE should be the same as number of atoms in the primitive cell. In the case of GaAs; SITE contain multiple ATOM tokens. POS= just next to ATOM is taken as subtokens under ATOM token. [8] In cases, we specify such subtokens as SITE_ATOM_POS.

In the SITE category, we place atoms (MT names) in the primitive cell. In these cases we use defaults atomic symbol (MT names) for `ATOM`. POS is in the Cartesian coordinate (in the unit of ALAT+DALAT).

To test ecalj, you may make a test directory and copy a ctrls.* to your directory. If you have VESTA and ecalj/StructureTool/ installed, you can see its structure by

```
$ viewvesta ctrls.cu
```

(here $ means command prompt).

> NOTE: As written in ecalj/README, you have to install VESTA and `viewvesta`. Then set VESTA= at the top of ecalj/Structure/viewvesta, and make softlink to it. The command `viewvesta`(`~/ecalj/StructureTool/viewvesta.py`) generate `POSCAR_cu.vasp` first, then send it to VESTA. `viewvesta` also accept `POSCAR_cu.vasp` directly. Except names starting from `ctrl` and `ctrls`, `viewvesta` sends the name to VESTA directly. We need extension '.vasp' to recognize it is written in VASP format. We have samples in `~/ecalj/StructureTool/sample`.
> A tool `vasp2ctrl` converts POSCAR_.vasp to ctrls.. "–help" show a small help.
> • ecalj/StructureTool/ is not tested well. Not believe it so much... We will fix it on your request.

In `ctrls.srtio3`, we use an expression 1/2 to give POS. We can use mathematical expression instead of values. Mathematical expressions such as "$+ - */$ sqrt(...)" are recognized. (instead of `3**2`, use `3^2`. Use parenthesis, and no space for an expression). We can use default atomic symbols (to check default atom name (MT name) type `ctrlgenM1.py --showatomlist`). Instead of such default symbols, we can use your own symbol as

```
SITE
  ATOM=M1 POS=1/2 1/2 1/2
  ATOM=M2 POS= 0   0   0
  ATOM=O  POS=1/2  0   0
  ATOM=O  POS= 0  1/2  0
  ATOM=O  POS= 0   0  1/2
SPEC
  ATOM=M1 Z=38
  ATOM=M2 Z=22
  ATOM=O  Z=8
```

[8] This may looks slightly uncomfortable since the end of range of ATOM is not clearly shown; it end just at the next ATOM token or new category.

. Then we have to add extra category SPEC where we set Z number. (You can use Z=37.5 for virtual crystal approximation, however, you can not do it in ctrls now. Edit it in ctrl file. Such a procedure will be explained in EcaljUsage.pdf.)

This is an example for Antiferro NiO:

```
#id  = NiO
%const bohr=0.529177 a=7.88
STRUC   ALAT={a} PLAT= 0.5 0.5 1.0  0.5 1.0 0.5  1.0 0.5 0.5
SITE    ATOM=Niup POS=  .0   .0   .0
        ATOM=Nidn POS= 1.0  1.0  1.0
        ATOM=O POS=  .5   .5   .5
        ATOM=O POS= 1.5  1.5  1.5
SPEC
    ATOM=Niup Z=28 MMOM=0 0  1.2 0
    ATOM=Nidn Z=28 MMOM=0 0 -1.2 0
    ATOM=O Z=8 MMOM=0 0 0 0
```

In this case, we define Niup and Nidn sites. These are recognized as Ni atom because of given Z number in SPEC. The subtoken MMOM=Ms,Mp,Md,Mf... are to specify number of magnetic moments (μ_B) for s,p,d,f channels (difference of up - down electrons within MT sites) as initial condition. In this case, we set n(up)-n(down)=1.2 for Niup site for d channel. Even just one ATOM name is given by yourself, all ATOM in SPEC should be given (in this case SPEC for O should be given).

We can see other samples in `~/ecalj/lm7K/TESTsamples/*/ctrls.*`. (we also have a sample generator. See later section.) Note that ctrls file is jut in order to generate default ctrl file in the followings. Not from ctrls but from ctrl, we can start calculations. (thus ctrls is not needed if we prepare ctrl file directory).

It is possible to add `RELAX= 0 0 1` after `SITE_ATOM_POS`; this means structure relaxation along z-axis (also need to set `DYN` category
as seen at `http://titus.phy.qub.ac.uk/packages/LMTO/tokens.html#DYNcat`), but its defaults are given (but commented out) automatically in the ctrl file generated by the procedure described in the following section). We detail it in EcaljUsage.pdf.

After `ctrl.*` is generated as shown below, we can run a quick command `lmchk` to check weather crystal structure is correctly given or not. And show symmetry information, and so on.

3.2 Generate default ctrl from ctrls by ctrlgenM1.py

To run programs of lm7K (lmfa, lmf, lmchk) in ecalj, we need an input file `ctrl.*`, which contains many other settings. To generate `ctrl.*` from `ctrls.*`, we have a command "ctrlgenM1.py" (written in python 2.x and call fortran code internally). Two steps required to complete ctrl file: (i) we give reasonable

options when we run ctrlgenM1.py. Then (ii) we may need to edit the ctrl file afterward.

At first, try `ctrlgenM1.py` without arguments. It shows help. To generate `ctrl` from `ctrl`, type

```
$ ctrlgenM1.py cu --nk1=8
```

Here cu specify ctrls.cu. The option –nk1=8 means the number of division of the Brillouin zone for integration. It means 8x8x8 division. If we like to use 8x8x4, we have to supply three arguments –nk1=8 –nk2=8 –nk3=4. The above command gives following console output.

```
$ ctrlgenM1.py cu --nk1=8
 === INPUT arguments (--help gives default values) ===
  --help  Not exist
  --showatomlist  Not exist
  --nspin=1
  --nk=8
  --xcfun=vwn   !(bh,vwn,pbe)
  --systype=bulk !(bulk,molecule)
  --insulator  Not exist !(do not set for --systype=molecule)

...

OK! A template of ctrl file, ctrlgen2.ctrl.cu, is generated.
```

As we see above, options which you specified are shown at the beginning of the console output (in this case –nk1=8). Others such as –nspin=1 are default settings. If we like to perform spin-polarized calculations, we add other option '–nspin=2' as

```
ctrlgenM1.py nio --nspin=2 --nk1=6
```

(NOTE: In the spin-polarized case, we need to set initial condition of size of magnetic moment at each atoms. Set it in `ctrls.*` as in the previous section, or edit MMOM of ctrl file (`MMOM=s p d f ...`) to be like `MMOM=0 0 1.2`.). The `ctrlgenM1.py` generates ctrl file named as `ctrlgenM1.ctrl.cu`. To do calculations, copy it to ctrl.cu so that lmf can recognize it.

```
cp ctrlgenM1.ctrl.cu ctrl.cu
```

3.3 crystal structure checker: lmchk

Do lmchk to confirm correct crystal structure is really given or not.

```
lmchk --pr60 cu
```

Then it reads `ctrl.cu`. `--pr60` is an option of verbose. Bigger number gives more information.

- Lattice info, Space group symmetry operations (in lmf format), and their generators (these operations can be generated from a few of them.) See `http://titus.phy.qub.ac.uk/packages/LMTO/tokens.html#SYMGRPcat+` about how to represent the operations.
- Show atomic positions in ctrl file.
- Tabulate MT radius and distance between atomic sites.

(lmchk –help shows help, but difficult to see. Not need to read it first.)

lmchk is also shows atom (MT site) id (position and class(equivalent positions). This is needed to interpret PDOS.

3.4 ctrl file

It is not necessary to look into ctrl file first, although some details are explained in the generated ctrl file. Please compare obtained results by lmf with those by other packages or literatures; let me know if you find something strange or your questions.

It is necessary to edit ctrl file to use full ability of lmf. For example, LDA+U, atomic position relaxation, core level spectroscopy, Change setting of default MTO and lo, better mixing procedure for stable convergence; higher accuracy, and so on.

But a few of ctrl file is easy to modify. Search these words and read explanations embedded in ctrl file.
(1)XCFUN (choice of XC—it is not need to repeat ctrlgenM1.py). It is also possible to change number of k points for sampling, to modify crystal structure slightly, and so on; all things needed are in ctrl. It is not needed to repeat ctrlgenM1.py again.
(2)SO (to obtain correct dispersion around top of valence at Gamma for GaAs, we need to set SO=1 and NSPIN=2; it is possible to run it as one-shot after convergence with `OPTIONS_Q=band`. and run by `lmf gaas --rs=1,0`).

`lmf --input` shows what can we write in ctrl file. But more than half are not for users, but for developers (or irrelevant now).

3.5 Do LDA/GGA calculations, and get convergence

Here we show how to get converged results from a `ctrl` file.

At first, we need initial guess of charge density. It can be given by a super position of atomic charge density. To obtain the charge density, we solve atoms first. It is by

```
$ lmfa gaas | tee llmfa
```

It takes just a few seconds. Here tee is a command of Linux. It keeps console output (standard output) to a file (llmfa in this case).

Then try

```
$ grep conf llmfa
```

. Then you see a key point that

```
conf:SPEC_ATOM= Ga : --- Table for atomic configuration ---
conf:  isp  l  int(P) int(P)z    Qval     Qcore   CoreConf
conf:    1  0       4  0         2.000    6.000 => 1,2,3,
conf:    1  1       4  0         1.000   12.000 => 2,3,
conf:    1  2       4  3        10.000    0.000 =>
conf:    1  3       4  0         0.000    0.000 =>
conf:    1  4       5  0         0.000    0.000 =>
conf:-----------------------------------------------------
conf:SPEC_ATOM= As : --- Table for atomic configuration ---
conf:  isp  l  int(P) int(P)z    Qval     Qcore   CoreConf
conf:    1  0       4  0         2.000    6.000 => 1,2,3,
conf:    1  1       4  0         3.000   12.000 => 2,3,
conf:    1  2       4  3        10.000    0.000 =>
conf:    1  3       4  0         0.000    0.000 =>
conf:    1  4       5  0         0.000    0.000 =>
conf:-----------------------------------------------------
```

This is an initial electron distribution, and how we divide core and valence. In this case core charge Qcore are (6 electron for s channel=1s,2s,3s and 12 electron for 2s and 3p). Core is not treated separately from valence electrons (frozen core approximation; we superpose rigid core density to make all-electron density). Qval means electrons for each s,p,d channels. The valence channels are 4s,4p,4d,4f (when we set EH=s,p,d,f). The int(P)z column is for local orbital. Thus we have 3d treated as local orbital. (ecalj can add one local orbital per l.)

isp means spin (1 or 2), since –nspin=1 for Ga and As, no isp=2 exist. In summary we have 4s,4p,4d,3d,4f as valence. This means we use corresponding number of MTOs and local orbitals.

After lmfa, let us start main calculation.

```
$ lmf cu
```

In unix, we ca save console output to llmf by `$ lmf cu | tee llmf`. As it starts iteration calculations, it shows similar output again and again (it is a little too noisy now). Then you end up with self-consistent result as

```
   ......
  it  8  of 30    ehf=   -3304.895853   ehk=   -3304.895853
 From last iter    ehf=   -3304.895856   ehk=   -3304.895855
 diffe(q)=  0.000003 (0.000007)    tol= 0.000010 (0.000010)   more=F
c ehf=-3304.8958531 ehk=-3304.8958529
 Exit 0 LMF
 CPU time:    7.024s     Mon Aug 19 02:03:19 2013   on
```

`it  8  of 30` means it stop at 8th iteration, although we set maximum number of iteration 30. Note that this number is given by `ITER_NIT=30` in `ctrl.cu`).

`ehf` and `ehk` are the ground state energy in Ry. They are calculated in a little different procedure. Although they are different during iterations, it finally get to be the almost the same number. (But they can be slightly different even converged for large systems. But you don't need to care it so much).
NOTE: ehk:Hohenberg-Kohn energy, ehf: Harris-Faulkner energy.

"grep diffe lllmf" shows how the changed of total energy (and charges) during iteration. diffe mean changes of energy with previous iteration, (q) is for electron density difference as well. See also save.* file, which only show ehk and ehk obtained by each iteration.

" grep gap llmf" shows how the band gap changes (in the usual setting), two same numbers per iteration are shown now.

Thus we do have ground state energy. Although output of lmf is long, most of all are to monitor convergence (we will shrink it). As long as it converged well, you don't need to look into it in detail. Eigenvalues are shown as

```
bndfp:  kpt 1 of 4, k=  0.00000  0.00000  0.00000   ndimh = 122
-1.2755 -1.2008 -1.2008 -0.2052 -0.2052 -0.2052 -0.0766 -0.0766 -0.0766
-0.0174 -0.0174 -0.0174  0.1094  0.1095  0.1095  0.2864  0.2864  0.4170
 0.4170  0.4736  0.6445  0.6445  0.6445
```

This is at k= 0.00000 0.00000 0.00000 . (because of historical reason, two same bndfp: are shown in each iteration; two band path method). "`lmf cu| grep -A6 BZWTS`" shows the Fermi energy (for insulator, we see band gap). Deep levels which gives little dependence on k are core like levels. These are in Ry; zero level is not so meaningful (for convenience, it is simply determined from the potential at MT boundaries).

rst.* contains is the main output which contains electron density. mix.* is a mixing file (which keeps iteration history). When you restart lmf again, it read rst.cu and mix.cu. If you start from lmfa result, please remove them. We can do parallel calculation with lmf-MPIK, we can invoke it with mpirun -np 8 lmf-MPIK cu. It should give the same answer.

3.6 DOS, Band, PDOS plot

We already have script to plot dos, band, and pdos from the result of lmf self-consistent calculations. We have scripts

```
job_tdos, job_band_nspin1, job_band_nspin2, job_pdos
```

. Look into these scripts, and then you see how to plot them.

For total DOS plot, it is better to check ctrl file; `BZ_TETRA=1`(this is default; thus make sure that `BZ_TETRA` do not exist or `BZ_TETRA=1`). In addition, it might be better to enlarge number of k point `NKABC` in ctrl file to have smooth curve. Then we do

```
job_tdos cu
```

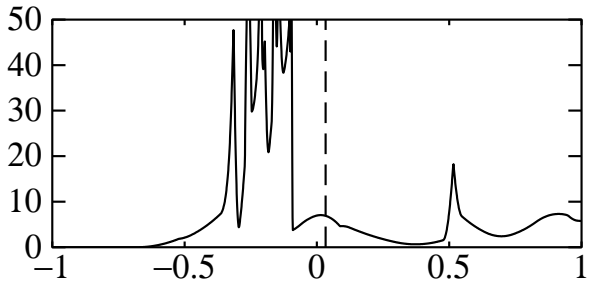

Figure 1: DOS(Cu)

This shows total DOS as

For band plot, we have to set symmetry lines along which we plot eigenvalues. Collections `syml.*` are in ecalj/MATERIALS/. Choose and modify one of them and rename it. I will gather other samples soon. 'BZ wikipedia' or something else will help you to interpret it.

To do band plot, we need `syml.cu` in your directory.

```
$ cp ~/ecalj/MATERIALS/Cu/syml.cu .
```

Then check `syml.cu`; it is

```
21  .5 .5 .5     0  0 0          L to Gamma
21   0  0  0     1  0 0         Gamma to X
0    0 0 0  0 0 0
```

First line means, we calculate eigenvalues for **k** points from **k**=(0.5,0.5,0.5) to **k**=(0,0,0). "L to Gamma" is just a comment since program only read seven numbers for each line. Second line means, we calculate eigenvalues for k points from **k**=(0,0,0) to **k**=(1,0,0). 3rd line means calculation just stop here. Units of **k** are in $2\pi/$`ALAT` (or $2\pi/$`(ALAT+DALAT)` if `DALAT` exist.). A line starting from '#' is neglected (comment line).

To do band plot, run

```
$ job_band_nspin1 cu
```

. This is for nspin=1. `job_band_nspin2` is for nspin=2 (These scripts try to determine the Fermi energy first. You may skip it in cases (but need to change the script)).

For PDOS plot,

```
job_pdos cu
```

It shows figures (number of figures are number of atoms in the cell) in gnuplot (they are written in the same position on X-window; move top one a little). The command `job_pdos` is a little time-consuming because we use no symmetry to distinguish all lm channels. (PDOS is not yet implemented for SO=1 case; spin-orbit coupling $\mathrm{L}\dot{S}$ is added.) We can edit script of gnuplot (pdos.site*.*.glt) for

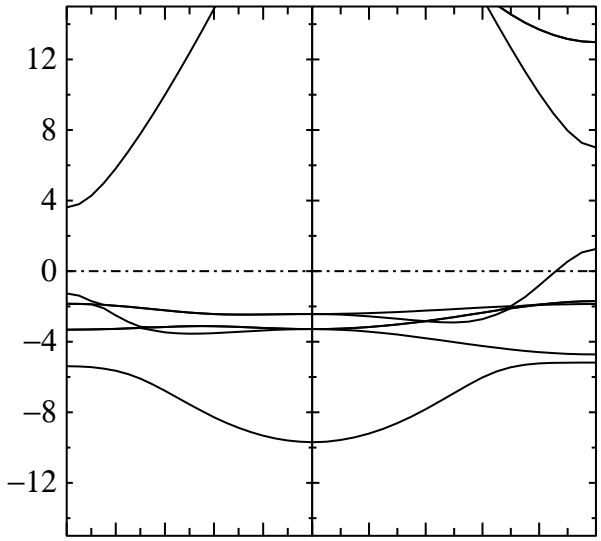

Figure 2: band plot(Cu)

your purpose. In principle, meanings of all data files are shown (see at the bottom of console output about lm ordering in a line), thus not so difficult to rewrite *.glt. For example, to plot eg and t2g separately. (NOTE: site id is shown by lmchk).

WARNING: Usually lmf and so on recognize options such as -v option. For example, ' lmf gaas -vnspin=2' or 'lmf gaas -vso=1'. This option changes values of variables defined in `% const` section. This is recorded in save.* file, and also shown at the top of console output. However, job_tdos and so on, do not yet accept these options. Thus we need to modify ctrl file without using -v option. Or you need to write these option to these command by hand (we will fix this problem in future.)

3.7 Useful samples: ecalj/MATERIAL/

Not only `ecalj/lm7K/TestSamples` (some of them are by older version), We have a material database in `ecalj/MATERIALS/`. Move to the directory, and type

```
$ ./job_materials.py
```

Then it shows a help. You see

```
...
=== Materials in Materials.ctrls.database are:===
  2hSiC  3cSiC  4hSiC  AlAs  AlN  AlNzb  AlP  AlSb  Bi2Te3  C
  CdO  CdS  CdSe  CdTe  Ce  Cu  Fe  GaAs  GaAs_so  GaN
  GaNzb  GaP  GaSb  Ge  HfO2  HgO  HgS  HgSe  HgTe  InAs
  InN  InNzb  InP  InSb  LaGaO3  Li  MgO  MgS  MgSe  MgTe
  Ni  NiO  PbS  PbTe  Si  SiO2c  Sn  SrTiO3  SrVO3  YMn2
  ZnO  ZnS  ZnSe  ZnTe  ZrO2  wCdS  wZnS
...
```

. For these simple materials (now 57 materials), input files can be generated, and run them automatically by a command `./job_materials.py` below. The ctrls are stored in `ecalj/MATERIALS/Materials.ctrls.database` in a compact manner (in addition, options passed to ctrlgenM1.py and options to lmf-MPIK are included). See `ecalj/MATERIALS/README` about how to add new material to it; it is not difficult. The command `./job_materials.py` gives ctrls.* for these materials from descriptions in the `Materials.ctrls.database`. And then it generates ctrl file by calling ctrlgenM1.py internally, and run lmfa lmf-MPIK successively (when no –noexec).

Try `./job_materials.py Fe --noexec`. (not fe but Fe as it shown above) at `ecalj/MATERIALS/`. Then it makes a directory Fe/ and set ctrl.fe (also ctrls.fe) in the directory. Without '–noexec', it does calculation for Fe successively. As for NiO and Fe, we see that `./job_materials.py` gives SPEC_ATOM_MMOM in generated ctrls and ctrl files. (Look into ctrls.fe; we need SPEC section when we add MMOM.)

Try `job_materials.py GaAs Si`.
Then directories GaAs/ and Si/ are generated. See `save.*` files containing total energies iteration by iteration. Starting from `ctrl.*` in these directory, the command perform DFT calculations (Console output is stored in `llmf`, `save.*` gives total energies. `rst.*` contains self-consistent density, from which we can calculate energy bands and so on).

"`./job_materials --all --noexec`" generates ctrls and ctrl files of these materials. "`./job_materials --all`" do self-consistent LDA calculations for materials (it takes an hour or more. To change the number of cores for lmf-MPIK, set option -np (number of core). See help of `./job_materials` (type this without arguments).

To make band plot and so on for Fe, follow instructions already explained.

```
  $ ./job_materials.py Fe  (and need to type return)
  (If you like start over, remove Fe/ under it first).
  $ cd Fe
  $ ./job_materials.py fe
    (but it might be better to do --noexec, and observe Fe/ctrls.fe and
Fe/ctrl.fe first. grep conf llmf shows the initial electron distribution).
  $ cat save.fe  (this shows total energies of each iteration. 'c ' at
```

```
the first column gives converged result. 'h ' is from atm file.)
        If it does not ends with 'c ...' line, something strange
occurs. see llmf (console out put of lmf is saved to llmf).
  $ cp ../syml.fe .
  $ job_band_nspin2 fe
        (As I said, this shell script do not yet accept
    options to lmf. Look into the script).
        (This calculate fermi energy first for safe; it takes
    some time)
  $ job_tdos fe
  $ job_pdos fe (as I said, this supress space-group symmetry, thus time consuming).
```

At the end of `job_pdos`, we show a help which pdos data is where (In pdos file, we have 26 numbers a line; first is energy, 2-26 are pdos for s,p,d,f,g; which is which are shown in the help). See joblmf file also (it contains options to invoke lmf. This is shown in save.*. In principle, options in joblmf should be passed to band plot and so on. But not yet implemented (it is not so difficult; I have to do it).

After doing `./job_materials` foobar, you may like move it back to original... In such a case, git works. At `ecalj/`, do

```
  $ mv MATERIALS MATERIALS.bk
  $ git checkout MATERIALS
```

Then you can see `MATERIALS/` is moved back to just downloaded one.

4 QSGW calculation

In the QSGW, we calculate a kind of *non-local exchange-correlation potential* $V^{\rm xc}(\mathbf{r},\mathbf{r}')$, by a procedure of GW calculation (it is quite time-consuming part). Then difference $V^{\rm xc}(\mathbf{r},\mathbf{r}')V - V_{\rm xc}^{\rm LDA}(\mathbf{r})\delta(\mathbf{r}-\mathbf{r}')$ is stored into `sigm` file. Then, we again do one-body calculation by lmf (or lmf-MPIK) where we add this sigm to one-body potential. Thus this means that we replace $V_{\rm xc}^{\rm LDA}(\mathbf{r})\delta(\mathbf{r}-\mathbf{r}')$ with $V^{\rm xc}(\mathbf{r},\mathbf{r}')V$. This iteration cycle is performed by a script "gwsc" as we explain later on. (In the default setting of `ctrl.*` file, lmf try to read sigm.* file as long as it exists. If not, do lmf or lmf-MPIK calculation. Note that `gwsc` makes a

4.1 GWinput

In order to perform QSGW, one another input file `GWinput` (no extension) is necessary in addition to `ctrl.*`. Thus all input files for QSGW is just two files, ctrl.* and GWinput. A template `GWinput` can be generated by a script `mkGWIN_lmf2`. You may have to modify it in cases for your purpose.

Let us start from ctrls.si;

```
#id  = Si
```

```
%const bohr=0.529177 a= 5.43095/bohr
STRUC
     ALAT={a}
     PLAT=0 0.5 0.5  0.5 0 0.5  0.5 0.5 0
SITE
     ATOM=Si POS=0.0 0.0 0.0
     ATOM=Si POS=0.25 0.25 0.25
```

. Do `ctrlgenM1.py si --tratio==1.0 --nk1=6` and copy ctrlgenM1.ctrl.si to ctrl.si. NOTE: the option `--tratio=1.0` means we use touching MT; this can be checked by `lmchk si`; since defaults is almost unity (`--tratio=0.97`), this is irrelevant, just to explain options.

We have to write `GWinput`. The default is given automatically by a command `mkGWIN_lmf2`;

```
$ lmfa si (lmfa is needed to do in advance).
$ mkGWIN_lmf2 si
......
== Type three integers n1 n2 n3 for Brillowin Zone meshing for GW! ==
 n1=
```

Then it pause and ask numbers. You have to type three numbers as 2+ return + 2+return+2 return.

```
== Type three integers n1 n2 n3 for Brillowin Zone meshing for GW! ==
 n1= 2
 n2= 2
 n3= 2
2 2 2
...(skip)...
OK! GWinput.tmp is generated!
```

Generated file is `GWinput.tmp`; you have to copy it to `GWinput`.

```
$ cp GWinput.tmp GWinput
```

These '2 2 2' you typed is reflected in a section 'n1n2n3 2 2 2 ' in `GWinput`. This means 2x2x2 (8 points in 1st BZ). You can edit it, and change it to e.g. 'n1n2n3 4 4 4' if you like to calculate self-energy on dense BZ mesh 8x8x8.

The template of GWinput is usually not so bad. But it may give a little expensive setting (or not very good enough in cases). Read EcaljUsage.pdf.

4.2 do QSGW calculation

Let us perform QSGW calculation. For this purpose, we use a script `gwsc`. We need to do `lmfa` in advance. Then do (not need to do `lmf`);

```
gwsc (number of iteration+1) -np (number of nodes) (id of ctrl)
```

If (number of iteration+1)=0, it gives one-shot calculation from LDA. But it is different from the usual one-shot; since it calculates off-diagonal elements of self-energy also, we can plot energy band dispersion. In cases (for usual semiconductors), it can give rather reasonable results in comparison with experiments from practical point of view.

This is an example of one iteration of QSGW cycle. (now a little different but essentially similar)

```
takao@TT4:~/ecalj/test1$ gwsc 0 -np 2 si
gwsc 0 -np 2 si
### START gwsc: ITER= 0, MPI size=  2, TARGET= si
--- No sigm nor sigm.$TARGET files for starting ---
 ---- goto sc calculation with given sigma-vxc --- ix=,0
No sigm ---> LDA caculation for eigenfunctions
        Start  mpirun -np 2 /home/takao/ecalj/TestInstall/bin/lmf-MPIK  si > llmf_lda
OK! --> Start  echo 0| /home/takao/ecalj/TestInstall/bin/lmfgw si > llmfgw00
OK! --> Start  echo 1|/home/takao/ecalj/TestInstall/bin/qg4gw > lqg4gw
OK! --> Start  echo 1|mpirun -np 2 /home/takao/ecalj/TestInstall/bin/lmfgw-MPIK  si> llmfg
OK! --> Start  /home/takao/ecalj/TestInstall/bin/lmf2gw >llmf2gw
OK! --> Start  echo 0|/home/takao/ecalj/TestInstall/bin/rdata4gw_v2  > lrdata4gw_v2
OK! --> Start  echo 1| /home/takao/ecalj/TestInstall/bin/heftet > leftet
OK! --> Start  echo 1| /home/takao/ecalj/TestInstall/bin/hchknw > lchknw
OK! --> Start  echo 3| /home/takao/ecalj/TestInstall/bin/hbasfp0 > lbasC
OK! --> Start  echo 3| mpirun -np 2 /home/takao/ecalj/TestInstall/bin/hvccfp0 > lvccC
OK! --> Start  echo 3| mpirun -np 2 /home/takao/ecalj/TestInstall/bin/hsfp0_sc > lsxC
OK! --> Start  echo 0|/home/takao/ecalj/TestInstall/bin/hbasfp0  > lbas
OK! --> Start  echo 0| mpirun -np 2 /home/takao/ecalj/TestInstall/bin/hvccfp0  > lvcc
OK! --> Start  echo 1|  mpirun -np 2 /home/takao/ecalj/TestInstall/bin/hsfp0_sc > lsx
OK! --> Start  echo 11|  mpirun -np 2 /home/takao/ecalj/TestInstall/bin/hx0fp0_sc > lx0
OK! --> Start  echo 2|  mpirun -np 2 /home/takao/ecalj/TestInstall/bin/hsfp0_sc  > lsc
OK! --> Start  echo 0|  /home/takao/ecalj/TestInstall/bin/hqpe_sc  > lqpe
OK! --> == 0 iteration over ==
OK! --> Start  mpirun -np 2 /home/takao/ecalj/TestInstall/bin/lmf-MPIK  si > llmf_gwscend.
OK! ==== All calclation finished for gwsc 0 -np 2 si ====
```

Here `echo (integer)` is readin in at the beginning of the code. To see it, please look into gwsc script (gwsc is at ecalj/fpgw/exec/ and copied to your bin/ by make install2). In anyway, this console output shows calculations finished normally.

Now we get rst.si and sigm.si file which contains (static version of) self-energy minims $V_{\rm xc}^{\rm LDA}$. What we did is the one-shot GW from LDA result; but note that we calculate not only diagonal elements but also off-diagonal elements.

We can write energy dispersion (band plot) in the same manner in LDA. To do it, we need rst.si, sigm.si, ctrl.si, QGpsi. (but QGpsi is quickly reproduced). After you have syml.si (e.g. in ecalj/MATERIALS/), Do

```
$ job_band_nspin1 si
```

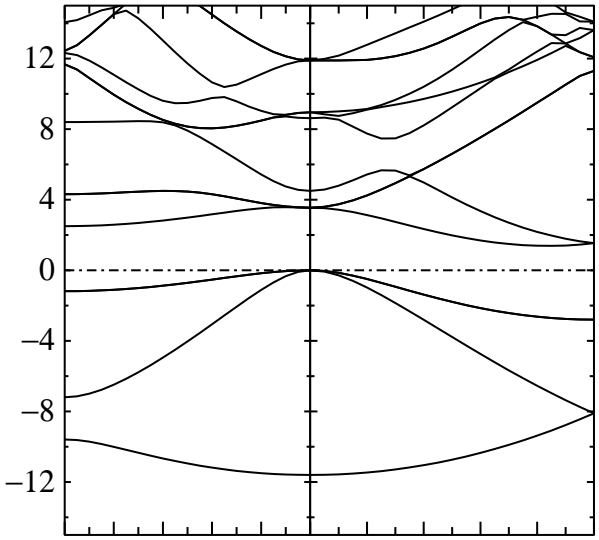

Figure 3: Si, one-shot GW with off-diagonal elements

You can observe large band gap as shown in the Fig.4.2. (To see it again, `gnuplot bnds.gnu.si -p`. All plots are in gnuplot, thus it is easy to replot it as you like).

We have QPU file (and also QPD for spin=2), which contains content of the diagonal part of self-energy. It will be explained elsewhere.

You can make total DOS and PDOS plot by

```
$ job_tdos si
$ job_pdos si
```

CAUTION:pdos plot is not allowed for so=1. (even tdos–¿ ask to t.kotani.)

To get final QSGW results, we have to repeat iteration until eigenvalues are converged. Note that total energy shown by console output llmf (and also shown in save file) is not so meaningful in the QSGW; we just take it as an indicator to check convergence. Let us repeat 5 iteration more. "-np 2" means one core to use.

```
$ gwsc 5 -np 2 si
### START gwsc: ITER= 5, MPI size=  2, TARGET= si
 --- sigm is used. sigm.$TARGET is softlink to it  ---
 ---- goto sc calculation with given sigma-vxc --- ix=,0
 we have sigm already, skip iter=0
 ---- goto sc calculation with given sigma-vxc --- ix=,1
 ...(skeip here) ...

OK! --> == 5 iteration over ==
OK! --> Start  mpirun -np 2 /home/takao/ecalj/TestInstall/bin/lmf-MPIK  si > llmf_gwscend.
```

```
OK! ==== All calclation finished for gwsc 0 -np 2 si ====
```

Note that we do skip 0th iteration (it is for one-shot from LDA) since we start from rst.si and sigm.si given by one-shot LDA. Thus we do just five iterations. Information of eigenvalues are in `QPU.(number)run` files. (for magnetic systems with nspin=2), wee have `QPD.(number)run` also). Check it by ls;

```
$ ls QPU.*run
QPU0.run  QPU.1run  QPU.2run  QPU.3run  QPU.4run  QPU.5run
```

(These are overwritten when we again repeat gwsc; be careful.) Note that `QPU0.run` was old one when you did 1-shot GW from LDA at the beginning. In anyway *.0run are confusing files; remove them).

In order to check convergence calculations going well during iteration, do

```
$ grep gap llmf*
```

This shows how band gap changes in llmf.*run files. In metal cases, we need to compare QPU file, magnetic moment or `grep '[xc] save.*`; this shows end of lmf iteration. Energy is not so meaningful but can be indicator to convergence.

Let us check convergence of the QSGW calculations. For this purpose, it is convenient to take a difference of QPU(QPD) files by a script `dqpu`. These files are human readable. To compare `QPU4.run` and `QPU5.run`, do

```
$ dqpu QPU.3run QPU.4run
```

Then we see a list of numbers (these are the differences of values in QPU files). Then it shows at the bottom as

```
Error! Difference>2e-2 between:   QPU.4run   and   QPU.5run
:  sum(abs(QPU-QPD))= 0.05736
```

but you don't need to care it so much. You rather need to check the difference of values. I can say most of all difference (especially around the Fermi energy are) are almost 0.00eV or 0.01eV, we can judge QPEs are converged. If not converged well, you may need to repeat `gwsc` again. (when the size of two QPU files are different, dqpu stops.)

5 new features

Wannier functions(under development). Recently T.Kotani includes the Wannier function generator, which was originally developed by T.Miyake and H.Kino on top of previous version of ecalj. Thus the Wannier functions (including effective interactions) can be generated in the PMT-QSGW.

6 How to add spin-orbit coupling

```
Do LDA and/or QSGW with SO=0 first.
```

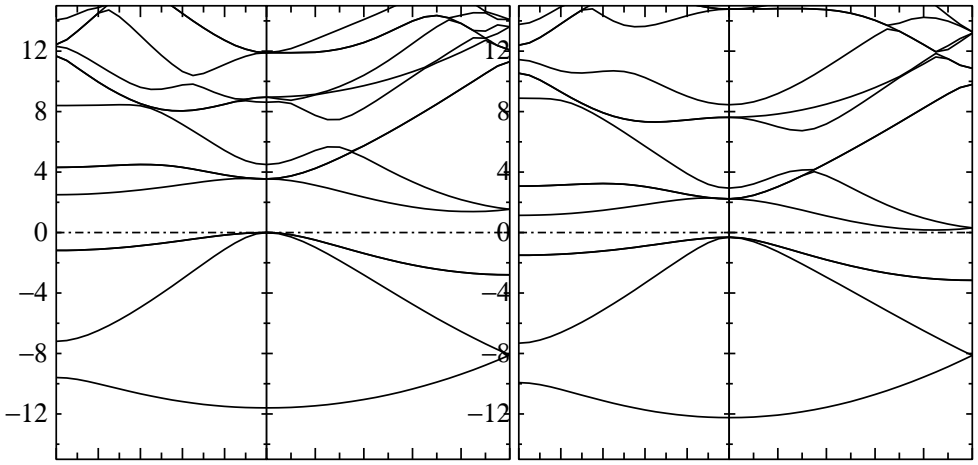

Figure 4: band plot(Si, QSGW one-shot test) and band(Si) (GGA)

```
Then apply the spin-orbit coupling by perturbation.

After converged with nspin=1 (or 2), create new directory and copy
  ctrl.gas, rst.gas, sigm.gas, QGpsi,
to it. Then we set
  nspin=2
  METAL=3 (usually this is default)
  SO=1  (this is ldots calculation off-diagonal elements included).
  Q=band (we do not change potential.)
in ctrl.gas.
Then run
>lmf gas >& llmf_SO
You can see "band gap with SO" by
> grep gap llmf_SO.
Then you can see two same lines.
  VBmax = 0.101949  CBmin = 0.236351  gap = 0.134402 Ry = 1.82786 eV
  VBmax = 0.101949  CBmin = 0.236351  gap = 0.134402 Ry = 1.82786 eV
(two lines per iteration is shown in metel mode).
This is the band gap with SO as a first-order perturbation
on top of the "QSGW without SO". When you use ctrl file generated by
ctrlgenM1.py. You can do the above procedure with
>lmf --rs=1,0 gas -vnit=1 -vso=1 -vnspin=2 -vmetal=3 --quit=band
(--rs=1,0 read rst.gas but not write rst.gas. Run lmf --help.
 The switch -v (-vso=1 in this case) replaces so=0 with so=1.
 This is recorded in save.gas file).

For band plot, you can use the same procedure
for the case without SO. (Look into the shell script job_band_nspin1.
```

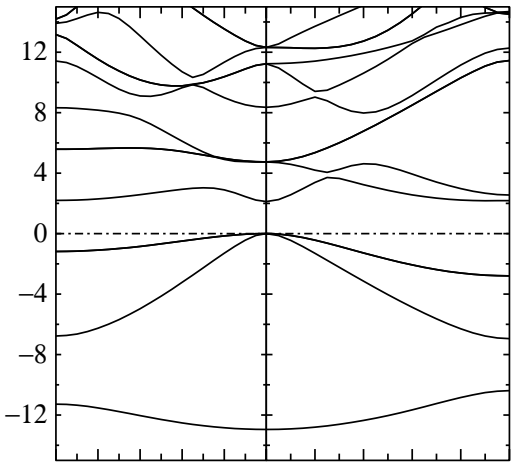

Figure 5: band(GaAs), QSGW (test case)

```
You have to modify it so that
  '--rs=1,0 gas -vnit=1 -vso=1 -vnspin=2 -vmetal=3 --quit=band'
is added to arguments for >lmf --band:syml ...).

(this will be simplified in future...)

---
For given sigm file, it seems possible to do self-consisent SO calculations
with keeping sigm (then we do not set Q=band).
However, note that Vxc is fixed in QSGW,
it is not necessary better than the above procedure.
```

References

[Kotani(2014)] T. Kotani, Journal of the Physical Society of Japan **83**, 094711 (2014).

[Kotani and van Schilfgaarde(2007)] T. Kotani and M. van Schilfgaarde, Physical Review B **76** (2007), 10.1103/PhysRevB.76.165106.

[Kotani and van Schilfgaarde(2010)] T. Kotani and M. van Schilfgaarde, Physical Review B **81**, 125117 (2010) [5 pages] (2010).

[Kotani and Kino(2013)] T. Kotani and H. Kino, Journal of the Physical Society of Japan **82**, 124714 (2013).

[Soler and Williams(1989)] J. M. Soler and A. R. Williams, Phys. Rev. B **40**, 1560 (1989).

[Soler and Williams(1990)] J. M. Soler and A. R. Williams, Phys. Rev. B **42**, 9728 (1990).

[Kotani and van Schilfgaarde(2002)] T. Kotani and M. van Schilfgaarde, Solid State Communications **121**, 461 (2002).

[Friedrich *et al.*(2010)Friedrich, Blügel, and Schindlmayr] C. Friedrich, S. Blügel, and A. Schindlmayr, Physical Review B **81** (2010), 10.1103/PhysRevB.81.125102.

==== reference is not yet completed... ====