

ecalj manual

<https://github.com/tkotani/ecalj>

February 23, 2015

Abstract

ecalj is an all-electron full-potential electronic structure calculation package available freely. This is based on the formulation in Refs. [Kotani et al.(2015)Kotani, Kino, and Akai] and [Kotani(2014)] (See Kotani2114QSGWinPMT.pdf and KotaniKinoAkai2015FormulationPMT.pdf at ecalj/Document/Manual/). By **ecalj**, we can do QSGW and related calculations based on the PMT=LAPW+LMTO method. To get minimum usage for ecalj, need to read up to Section .5 or Sec.6. (We can generate model Hamiltonian through Wannier functions; not documented well yet...)

Be careful. This document still contain many bugs... We need your help on this manual.

Requirement

To support of our ecalj activity, we need your acknowledgment to **ecalj** in your publications in the manner shown in ecalj/README.md;

<https://github.com/tkotani/ecalj/#we-need-acknowledgment-as> such as:

[1] ecalj package at <https://github.com/tkotani/ecalj/>.

Its one-body part is developed based on Ref.[2].

[2] LMsuit package at <http://www.lmsuite.org/>.

Its GW part is adopted mainly from Ref.[1].

(as in the same manner of usual papers).

Relation with LM suit at <http://www.lmsuite.org/>:

There is a full potential LMTO package (with a slightly different implementation of PMT) contained in **lmsuit**. In the package, there is the driver to use a modified version of the GW code in **ecalj** (GW part in **ecalj** is occasionally adopted in **lmsuit**). The FP-LMTO in **lmsuit** and **ecalj** have some similarities (for input files and outputs). Thus some part of its document in the web site is still applicable even to the ecalj package.

Contributions

Main contributors to the ecalj package are (only a list of people who directly related to the code developments):

T.Kotani, Hiori Kino, Mark van Schilfgaarde, Sergey Faleev, Takashi Miyake,

Seungwoo Jang, Manabu Usuda. They also contribute to this documents.

T.Kotani learned the LMTO-ASA-GW code by F.Aryasetiawan, which is on top of the Stuttgart's LMTO-ASA, mainly by van Schilfgaarde under O. Jepsen and O. K. Andersen

<http://www2.fkf.mpg.de/andersen/LMTODOC/LMTODOC.html>. FP-LMTO part is originally given by M.Methfessel, Mark van Schilfgaarde and their collaborators.

Contents

1	Introduction	4
1.1	Uniqueness of the ecalj package.	5
1.1.1	What do we expect for QSGW?	6
1.2	Rule in this manual	7
1.3	What can we do with the ecalj package?	7
2	Install	8
2.1	Binaries and Scripts	8
2.2	tests	8
2.3	Directory structure	9
3	LDA/GGA calculations and Plots	10
3.1	Write crystal structure file, ctrls	10
3.2	Generate default ctrl from ctrls by ctrlgenM1.py	13
3.3	crystal structure checker: lmchk	13
3.4	ctrl file	14
3.5	Run LDA/GGA calculations, and get convergence	14
3.6	DOS, Band, PDOS plot	16
3.7	Useful samples: ecalj/MATERIAL/	17
3.8	How to add spin-orbit coupling?	18
4	How to run QSGW calculation?	19
4.1	GWinput	20
4.2	Run gwsc script	21
5	gwsc script to perform QSGW	24
5.1	outputs of gwsc	24
5.2	Preparation stage of gwsc	25
5.3	Main stage of gwsc	25
5.4	Other functions (or scripts)	26
6	Cautions for usage	27
6.1	lmf -help	32
7	Wannier function	32
7.1	lwmatK1 and lwmatK2	33
8	ctrl file details	33
8.1	How to set local orbitals	36
9	GWinput details	39
9.1	generate a template of GWinput	39
9.2	overview of GWinput	39
9.3	General section	41
9.4	<QPNT> section	44
9.5	set QPNT for eps mode (QforEPS section)	46
9.6	<PRODUCT_BASIS> section	47
9.7	ANFcond (we can skip here since we do not check this option now. Need fix this if necessary. 49	
10	Main Output Files of GW part	51
10.1	QPU	51
10.2	XCU	51
10.3	SEXU	51
10.4	SEXcoreU	51
10.5	SECU	52
10.6	TOTE.UP (TOTE.DN)	52

10.7	TOTE2.UP (TOTE2.DN)	52
10.8	DOSACC.Ida	52
10.9	DOSACC2.Ida	52
10.10	Core.ibas*_l*.chk	52
10.11	VXCFP.chk	52
10.12	The Fermi energies in this <i>GW</i> code.	52
11	mkGWIN_lmf2 and its I/O Files	54
11.1	echo 0 lmfgw	54
11.2	gwinit	54
11.3	echo -100 qg4gw	55
12	gwsc script and its I/O Files	56
12.1	echo 0 lmfgw si	57
12.2	echo 1 qg4gw	57
12.3	echo 1 lmfgw si	57
12.4	lmf2gw	58
12.5	rdata4gw_v2	58
12.6	echo 1 heftet	59
12.7	hchknw	59
12.8	echo 3 hbasfp0	59
12.9	echo 0 hvccfp0	60
12.10	echo 3 hsfp0	60
12.11	echo 0 hsfp0	60
12.12	echo 1 hsfp0	60
12.13	echo 11 hx0fp0	60
12.14	echo 12 hsfp0	61
12.15	echo 0 hqpe	61
13	Check list for convergence on GW calculations	62
14	Linear response calculations	64
14.1	eps_lmfh, epsPP_lmfh: the dielectric functions	64
14.2	epsPP_lmfh: the dielectric function(No LFC— faster)	65
14.3	How to calculate correct dielectric function?	65
15	Used files	70
15.1	@MNLA_CPHI	70
15.2	CPHI and GEIG	71

•Reference

1 Introduction

The **ecalj** is an first-principles electronic-structure calculations package with some unique features. With **ecalj**, we can do not only standard calculations (LDA/GGA/LDA+U, relaxation of atomic positions), but also the quasiparticle self-consistent GW calculations(QSGW), linear responses (charge and spin), Wannier functions (and U of them).

This is base on an unique one-body problem solver, the PMT method (=the Linearized APW+MTO method) [Kotani et al.(2015)Kotani, Kino, and Akai]. Thus we identify the QSGW method implemented in **ecalj** as the PMT-QSGW method. Introduction to the PMT-QSGW is given in Sec.1.1. Today “QSGW” is accepted as a standard procedure in the electronic structure calculations [Bruneval and Gatti(2014)].

First, see README.md shown at <https://github.com/tkotani/ecalj#ecalj-> (or [ecalj/README.md](#) in the package). Free to download **ecalj** package from it, and use it. The QSGW code is version controlled by git.

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, main 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 compiler); 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 POSCAR to ecalj, and so on. PMT-QSGW shows more stable convergence than the previous version, FP-LMTO-GW ¹.

ecalj package mainly consists of two parts. One is one-body part(PMT part) (in [ecalj/lm7K/](#)), the other is many-body part(GW part) (in [ecalj/fpgw/](#)).

One-body part (PMT part), based on [Kotani et al.(2015)Kotani, Kino, and Akai]
We can perform standard calculations such as LDA,GGA,LDA+U, atomic position relaxation, and so on. In addition, the PMT part has an interface to perform GW (and QSGW) calculation: the one-body part can include a given non-local exchange-correlation potential stored in a file **sigm.***. ². The QSGW calculation is performed by a script **gwsc**, which has an iteration loop calling the one-body program (**lmf**) and many-body part (GW part) alternatively. The many-body part generate the file **sigm.***. See Fig.6 and around.

Many-body part (GW part), based on [Kotani(2014)]

As the inputs for the GW calculation, we have to supply the eigenfunctions and the eigenvalues from the one-body part to the GW part. The eigenfuncions re-expanded by the two types of basis functions, the atomic-like argumentation functions in the muffin-tin(MT) spheres, and the plane-waves in the interstitial region, say, the interstitial plane-wave (IPW) hereafter. IPW is defined as the usual plane waves in the interstitial region, but zero within MTs’. The IPWs + ”atomic like functions within MTs” make “a basis set to expand eigenfuncitons”. See Eq.(17) of [Kotani(2014)].

We need another basis set to expand “product of eigenfuncions”. That is, the mixed product bases (MPB)

[Kotani and van Schilfgaarde(2002), Friedrich et al.(2012)Friedrich, Betzinger, Schlipf, Blügel, and Schindlmayr], which consists of the two kinds of bases (caution: do not mixed up with the basis set for eigenfuncitons); (i)the local atom-centered functions confined to MT spheres, so-called the product basis; (ii) IPW. The product-basis are calculated from products of solutions to the Schrödinger equation within the MT sphere. The Coulomb matrix v , the dynamically screened Coulomb interaction W , and so on, are expanded in the MPB. It can virtually span all the space made

¹In 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.

²In the case of using **sigm.***, total energy shown in the console output (also in **save.***) are just the indicator, not the meaningful total energy

of product of eigenfunctions (but, in practical calculations, we need to use a small size of the bases to reduce computational time). We include full energy-dependence of $W(\omega)$. See Sec.3 of [Kotani(2014)].

Recently T.Kotani includes the Wannier function generator, which was originally developed by T.Miyake and H.Kino on top of previous version of GW part. Thus the Wannier functions (including effective interactions) can be generated in the PMT-QSGW. (U parameters in the full-screening and cRPA).

1.1 Uniqueness of the ecalj package.

We will explain two unique points of **ecalj**.

PMT:

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 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 et al.(2015)Kotani, Kino, and Akai, Kotani and van Schilfgaarde(2010)] 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's), to represent the eigenfunctions (no other methods use two kinds of augmented waves together). Thus eigenfunctions are expanded in the linear combinations of the APWs, MTOs, and the lo's. The formulation is clarified in Ref.[Kotani et al.(2015)Kotani, Kino, and Akai]. Then the electron density and the one-body potential are given in the "3-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 [Soler and Williams(1989), Soler and Williams(1990)]) is also used in the projected augmented wave (PAW) method such as VASP.

We now usually use highly localized MTOs together with APWs of low energy cutoff ($3 \sim 4$ Ry).³ I think this is promising not only for efficient DFT/QSGW scheme, but also for kinds of applications in future.

QSGW:

In **ecalj**, we can perform the GW calculation. The usual GW approximation is so-called "one-shot GW" starting from LDA. It usually only calculates 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 may fail when its starting point (eigenfunctions and eigenvalues supplied by LDA) is problematic. This is the reason why we originally develop the QSGW method. The QSGW now becomes popular and taken as a possible candidate to go beyond current limitation of such GW and LDA/GGA [Bruneval and Gatti(2014)]. 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 calculation⁴.

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

³current implementation have not yet efficiently use this locality; this must allow us to speed up one-body problem solver.

⁴Exactly speaking, we use LDA idea for efficient implementation of QSGW; thus obtained results are slightly dependent on the choice of LDA or GGA

1.69 eV ⁵. On the other hand, the QSGW predicts about 1.8~1.9eV, a few tenth of eV larger than experiment (for practical use, 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 starting 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 calculations are very time-consuming; 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.

1.1.1 What do we expect for QSGW?

Let us recall hybrid functionals such as B3LYP, and LDA+U. In hybrid functional methods, we use $V_{xc} = (1-\alpha)*LDA + \alpha*(\text{Fock exchange like term})$, where α is taken to be ~ 0.25 usually ⁶. The α can be dependent on materials; for metals α should be almost zero. For larger band gap insulator, α becomes larger. ⁷ 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 in LDA). 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 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. 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 $0.25*(\text{Fock exchange})$).

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.

⁵We undo electron-phonon effect (0.06eV) and spin-orbit effect (0.11eV) from the true the experimental value 1.52 eV.

⁶exactly speaking, we have range cutoff for Fock exchange term in the HSE functional in addition

⁷If you use $\alpha = 1$ (Hartree-Fock limit), the band gap of Si becomes 20eV or something.

The main purpose of QSGW is to determine an one-particle effective Hamiltonian H^0 ⁸, which describes the quasiparticle picture (or independent-particle picture) for the system we calculate. In other words, QSGW divides the 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.

1.2 Rule in this manual

- **This font** is for executable file(program) or shell scripts.
- **echo 3|hbasfp0** means doing **hbasfp0** with the argument '3' supplied as the standard input (read(*,*) in fortran).
- **This font** is for files, directories, contents of files, or variables used in codes.
- **ctrl.si,rst.si** and so on mean the case of Si. You may need to replace the extension **si** for your case. (this extension is given by user. Lower case, number, and underscore [a-z0-9_] are allowed.) In the followings, **ctrl.*** means a file wish such an extension.
- There are files named **foobarU** and **foobarD**, which are for up spin (isp=1) and for down spin (isp=2), respectively; for example, **SEXU** and **SEXD**. We sometimes use **foobarU** to denote **foobarU** and **foobarD** together.
- **k** vector in the Brillouin zone is called as **q** or **k**.

1.3 What can we do with the ecalj package?

At Feb.2015, what we can do is as follow. We have limited parallelization. (e.g. k point parallel).

- LDA/GGA LDA+U, calculations, atomic forces and relaxation. Spin-orbit is included only for co-linear spin-density cases.
- Quasi-particle(QP) energy in the 1st-iteration from LDA. (one-shot *GW*)
Make band plot for LDA and the QP energies.
- Spectrum function of the self-energy Σ . Life time (imaginary part) of QPs.
- Dielectric function, and its inverse. (including local-field effect or not).
- QP self-consistent *GW*(QSGW)
- magnetic susceptibility.
- Wannier function. (not only one-body part, but also effective interaction W and cRPA)

⁸people often pronounce this “ H-naught ”

2 Install

Install and minimum tests are easy; even in a note PC, e.g., we can use gfortran in Ubuntu 14.04 on e.g., Thinkpad T420s for a test purpose. For productive runs, we may need multi-cores. Current implementation for parallelization by MPI is limited (not so much especially for the dielectric function part yet). Thus, probably, it may be not so efficient to use too many cores.

Follow the instruction of `ecalj/README.md`.

or we can see the same one at <https://github.com/tkotani/ecalj/#install-and-test>. We have a command `ecalj/InstallAll.ifort` and so on. This command installs `ecalj` and run a series of install tests automatically.

2.1 Binaries and Scripts

Binary and Scripts contained in `ecalj` are

- **ctrlgenM1.py**
Generate default input file `ctrl.*` from the structure file `ctrls.*`. The latter file only contains information of crystal structure.
- **lmfa**
Spherical atom calculation as initial condition, and core charge
- **lmchk**
Check atomic positions, crystal symmetry, and computational conditions.
- **lmf** and **lmf-MPIK** LDA/GGA, LDA+U calculations. (or we can use `Vxc` in QSGW instead). We mainly use **lmf-MPI** (k-parallel version) instead of **lmf**.
- PROCAR mode of lmf: Fat band mode.
`mpirun -np 4 lmf-MPIK --mkprocar --band:fn=sym1 mgo` gives PROCAR
(Try an example `/ecalj/MATERIALS/MgO-PROCAR/`. Run `./job` at the directory).
- **gwsc**
QSGW calculation
- `job_band_nspin1, job_band_nspin2, job_fermisurface, job_tdos, job_pdos`
band, fermi surface, tdos, pdos plot.
- **epsPP_lmfh**
Dielectric function without local field correction (LFC).
- **eps_lmfh**
Dielectric function with LFC
- **epsPP_lmfh_chipm**
Non-interacting transverse spin polarization.
- **gw_lmfh**
One-shot GW calculation. This also show life-time of QPs (QPU_lmf). (we need make it parallelized...)
- **genMLWF**
Wannier functions and matrix elements of W on it. A implementation of cRPA included.
- **dqpu**
A small python script to compare QPU.* files (eigenvalues are compared) numerically. (Seungwoo say it cause "index error"; Need to fix dqpu if something strange).

2.2 tests

`Install.ifort` run tests at `ecalj/TestInstall`.

In the following, `si:gw_lmfh` means `'>make si_gw_lmfh'` at `ecalj/TestInstall/`; this test is performed with the Makefile at the directory.

si:gw_lmfh/	Results: QPU
si:gWSC/	: QPU,log.si
gas:gWSC/	: QPU,log.gas
nio:gWSC/	: QPU,log.nio
fe_epsPP_lmfh_chipm:	: ChiPM*
gas:eps_lmfh/	: EPS*
gas:epsPP_lmfh/	: EPS*

(These are just samples; not for practical calculations)

2.3 Directory structure

```

ecalj
|----- InstallAll.ifort, Install.gfortran ! install and test
|----- Document/
|----- fpgw/          ! full potential GW code
|----- fpgw/Wannier  ! Miyake's Wannier code is reorganized here.
|----- lm7K/         ! PMT method
|----- MATERIALS/   !job_materials.py contains examples.
|----- StructureTool/ ! POSCAR converter, and a utility to
|----- TestInstall/  ! Install test; this is invoked from Install.*
|----- TOOLS/        !Tools for developers

```

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.*` (See `ecalj/StructureTool/README.txt` for the tool. we have **vasp2ctrl** and **ctrl2vasp**. Type them without arguments to see help.)

2. Generate `ctrl.*` from `ctrls.*` by a script `ctrlgenM1.py`,⁹

Here `ctrl.*` is the main control file which contains all required information to perform calculations. `ctrl.*` contains not only the content of `ctrls.*`, but also other information needed for calculations. 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.

CAUTION for a known bug! : If a crystal structure is only slightly different from a structure with higher symmetry, the **lmchk** may give a wrong crystal symmetry. In such a case, you have to “standardize structure” by VESTA or some other tools. This occurs e.g., when we use a structure numerically relaxed by VASP.

4. Run **lmfa** (calculations of spherical atoms (MT sites) 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). It finished within a second.
5. Run LDA/GGA calculations. We can run the LDA/GGA calculation by **lmf** or **lmf-MPIK** (-MPIK means kpoint parallel version).
6. Post processing.

Plot energy band, DOS, PDOS, by running scripts. We can use scripts **job_band_nspin1** or **job_band_nspin2** for band plot (need `sym1.si` file (symmetry line for band plot)). We also have **job_pdos**, **job_tdos**, **job_fermi** and so on for DOS, PDOS, fermi surfaces. 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
```

⁹`ctrlgenM1.py` exists originally at `ecalj/lm7K/` (`ctrlgenM1.py` was already copied to your `BINDIR`= defined in `ecalj/Install.ifort` in the installation).

```

        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 **ctrls.***.¹⁰

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.

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.¹¹ 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 ctrl.s.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 ctrl.s, **viewvesta** sends the name to VESTA directly. We need extension '.vasp' to recognize it is written in VASP

¹⁰For 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.

¹¹This 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.

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. Another possible way is using `cif2cell`.

If you have a `cif` file, run

```
cif2cell foobar.cif -p vasp --vasp-cartesian --vasp-format=5
```

And convert `POSCAR` to `ctrls`. `cif2cell` is available from github.

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
```

. 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 other place.xxx)

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...` re 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/LMT0/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 other place xxx.

After `ctrl.*` is generated as shown below, we can run a command `lmchk` to check whether crystal structure is correctly given or not. It finish in a second. It show symmetry information, and so on used in the calculation.

CAUTION!: Positions of atoms are not necessarily fixed by `ctrl.*` when you restart calculation with `rst.*` file, because atomic positions are read from `rst.*`. We need to pay attention when we use DYN option because `lmf` run may save relaxed atomic positions into the `rst.*`. As “`lmf -help`” shows, `lmf si --rs=1,1,1,0,0` can read atomic position from `ctrl` file.

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 not only structures but also other settings. To generate `ctrl.*` from `ctrls.*`, we have a command “`ctrlgenM1.py`” (written in python 2.x and call fortran programs(`lmfa`,`lmchk`) internally). Two steps required to complete `ctrl` file: (i) we give reasonable options when we run `ctrlgenM1.py`. Then (ii) we may edit the `ctrl` file afterward. In anyway, `ctrl` file is the starting point of calculations; `ctrls` is required just in order to generate `ctrl`.

At first, try `ctrlgenM1.py` without arguments. It shows help. To generate `ctrl` from `ctrls`, 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 $8 \times 8 \times 8$ division. If we like to use $8 \times 8 \times 4$, 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 that we can let `lmf` know correct crystal structure. It also show crystal structure information, equivalent sites, site index and so on.

```
lmchk --pr60 cu (--pr# gives more informations if # is number)
```

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.

CAUTION for a known bug! See 3 in Sec.3.:

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 Γ point for GaAs, we need to set SO=1 and NSPIN=2. QSGW calculation (by **gwsc**) do not allow this option now; Thus we run **lmf** (or **lmf-MPIK**) with such settings changed in ctrl file, after QSGW is converged.

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 Run 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  1  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  1  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 Q_{core} are “6 electron for s channel=1s,2s,3s and 12 electron for 2s and 3p”. Q_{core} is treated by frozen core approximation. See Sec.2.5 in Ref.???. Q_{val} means electrons for valence s,p,d channels. The valence channels are 4s,4p,4d,4f (if we set EH=s,p,d,f) in this case The int(P)z column is for local orbital. Thus we have 3d treated as local orbital. (ecalj allow add one local orbital per l .)

The isp index means spin (1 or 2), since $-n\text{spin}=1$ (when we invoke ctrlgenM1.py) for GaAs, no $\text{isp}=2$ 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 can 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 llmf`” 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 `ehf` 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
```

This shows total DOS as The range of DOS and division of total DOS is given by DOSMAX

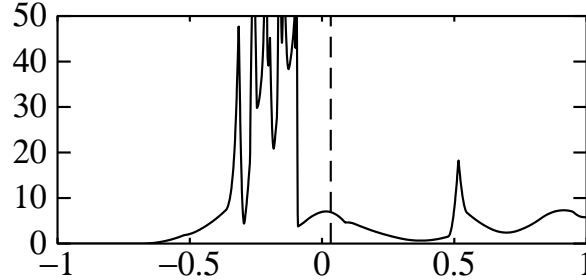


Figure 1: DOS(Cu)

and NPTS in ctrl. Edit tdos.cu.glt for gnuplot for your presentation. Please look into job_tdos file in your bin directory. It is a small script.

For band plot, we have to set symmetry lines along which we plot eigenvalues. Collections sym1.* 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 sym1.cu in your directory.

```
$ cp ~/ecalj/MATERIALS/Cu/sym1.cu .
```

Then check sym1.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 \mathbf{k} points from $\mathbf{k}=(0.5,0.5,0.5)$ to $\mathbf{k}=(0,0,0)$. "L to Gamma" is just a comment since program only read seven numbers for each line. Second line means, we calculate eigenvalues for \mathbf{k} points from $\mathbf{k}=(0,0,0)$ to $\mathbf{k}=(1,0,0)$. 3rd line means calculation just stop here. Units of \mathbf{k} are in $2\pi/\text{ALAT}$ (or $2\pi/(\text{ALAT}+\text{DALAT})$ 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 $L\hat{S}$ is added.) We can edit script of gnuplot (pdos.site*.glt) for your purpose. To plot again, run

```
gnuplot -persist pdos.site001.cu.glt
```

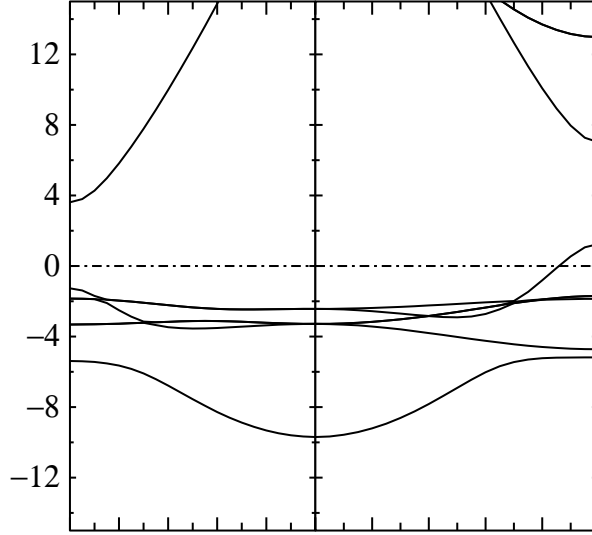



Figure 2: band plot(Cu)

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 `jobllmf` file also (it contains options to invoke `lmf`. This is shown in `save.*`. In principle, options in `jobllmf` 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.

3.8 How to add spin-orbit coupling?

Do LDA and/or QSGW with `S0=0` first.

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 fil e 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.
 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:sym1 ...).
 (--quit=band is not necessary if we like to renew eigenfunctions
 self-consistent. Anyway we expect little differences.)

For given sigm file, it seems possible to do self-consistent 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.

4 How to run QSGW calculation?

In the QSGW, we calculate a *non-local exchange-correlation potential* $V^{xc}(\mathbf{r}, \mathbf{r}')$, by a procedure
 of GW calculation (very time-consuming part). Then difference $V^{xc}(\mathbf{r}, \mathbf{r}')V - V_{xc}^{LDA}(\mathbf{r})\delta(\mathbf{r} - \mathbf{r}')$
 is stored into sigm.* file. The potential file sigm is a key to perform QSGW calculations as
 seen in Fig.6. The sigm contains static non-local potential $\Sigma_{QSGW} - V_{xc}^{LDA}$.

Then, we again do one-body calculation by **lmf** (or **lmf-MPIK**) where we add this sigm
 to one-body potential ; when we run **lmf** or **lmf-MPIK**(k-parallel mpi version), **sigm** is read
 and added to the one-body potential if we have HAM_RDSIG=12 in ctrl.*. Thus this means
 that we replace $V_{xc}^{LDA}(\mathbf{r})\delta(\mathbf{r} - \mathbf{r}')$ with $V^{xc}(\mathbf{r}, \mathbf{r}')$.

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.

To start QSGW calculation by **gwsc**, we need not only ctrl.si, but also another input file
 GWinput. Its template GWinput.tmp can be generated by the Step.2 as follows.

As a summary, you have to follow steps below in order to perform QSGW calculation.

Step 1. Perform from the Step.1 thru the Step.4 (up to `lmfa`) in Sec.3.5. (as same as the case of LDA/GGA).

You don't need to perform LDA calculation in advance, since `gwsc` perform LDA/GGA calculation at its beginning. (It means that we start from the one-body Hamiltonian H_0 in LDA/GGA as initial condition. In cases, LDA/GGA give poor initial conditions for QSGW; in such a case, we may need another trick to prepare starting point.).

[Caution: We have to use the same `LMXA` (l in the expansion of eigenfunctions in each MT) for all the MT spheres. (This is due to historical reason; we may need to fix this.)]
xxx need to explain LMA xxx

Step 2. Run the script `mkGWIN_lmf2`.

The purpose of this script is to get `GWinput.tmp`. Other files generated are not used in the following stage.

Step 3. Edit `GWinput.tmp` and save it as `GWinput`.

`GWinput` is the input file describing the computational conditions for *GW* calculation. Usually, the default setting gives reasonable results. To reduce computational time, we may use `lcutmx(atom)=2` for oxygen sites (this may be also other small atoms.). `grep lcut GWinput -A1` shows `lcutmx` for each atomic sites) These step 2. and step 3. are just only to get `GWinput`.

Step 4. Run the script `gwsc`.

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 Brillouin 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 Brillouin 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).

4.2 Run gwsc script

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> llmfgw01
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.0
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_{xc}^{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

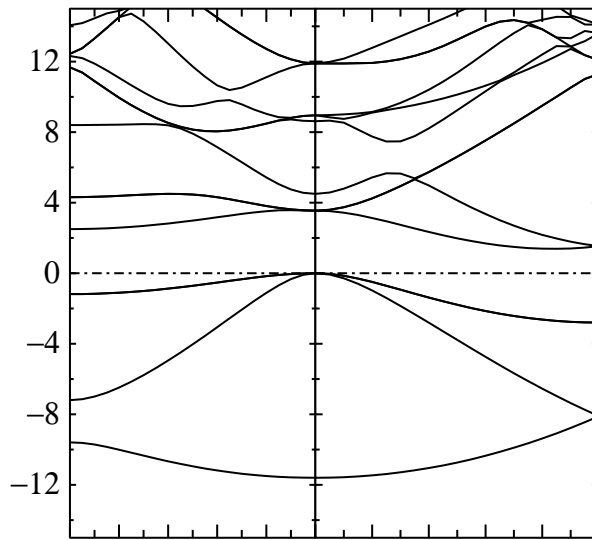


Figure 3: Si, one-shot GW with off-diagonal elements

```
$ job_band_nspin1 si
```

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-*i* 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/llmf-MPIK si > llmf_gwscend.0`

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.)

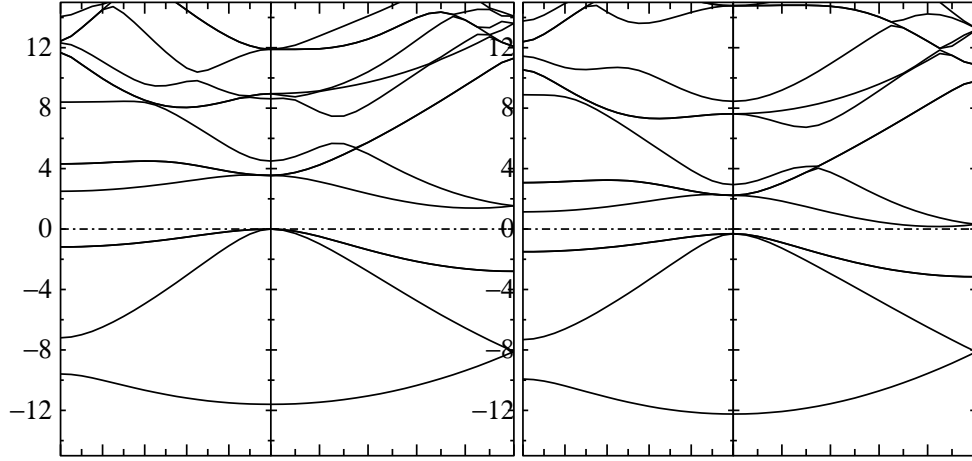


Figure 4: band plot(Si, QSGW one-shot test) and band(Si) (GGA)

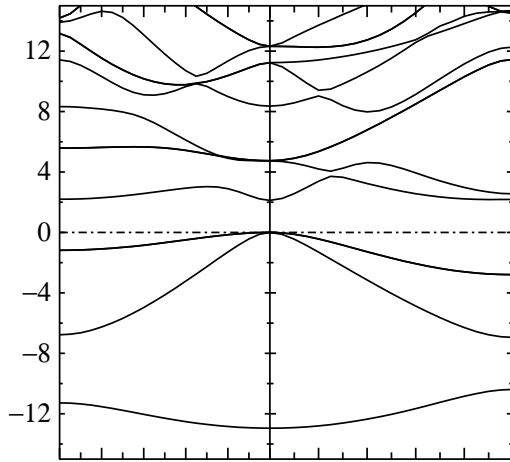


Figure 5: band(GaAs), QSGW (test case)

5 gwsc script to perform QSGW

5.1 outputs of gwsc

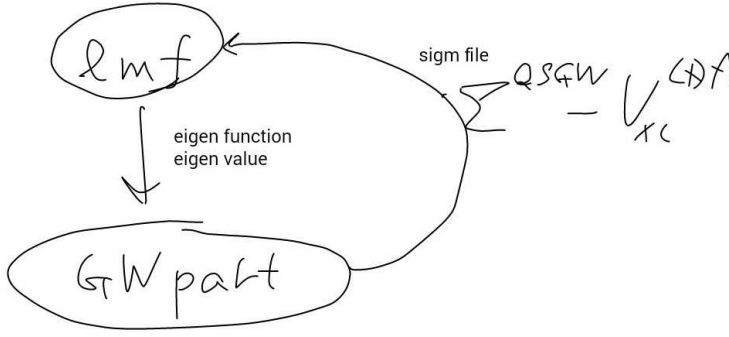


Figure 6: Shell script `gwsc` to perform QSGW contains an iteration loop to make `sigm` (and eigenvalues, eigenfunctions) self-consistent. The iteration loop is written in `ecalj/fpgw/exec/gwsc` (a bash script). Exactly speaking, we have to pass all the required data (not only eigenvalues and eigenfunctions, but also crystal structure, $\mathbf{q}+\mathbf{G}$ vectors, symmetry information and so on) to `GW part`. The purpose of the `GW part` is to calculate $\Sigma_{\text{QSGW}} - V_{\text{xc}}^{\text{LDA}}$.

When `gwsc` runs normally, it gives console output as follows. This is a case for `ctrl.gaas` for

```
>gwsc 10 -np 24 gaas
. Without arguments, typing gwsc shows usage as
>An example of usage: gwsc 5 -np 4 si, where 5 means 5+1 iterations
. We recommend you do look into the script gwsc. It uses run_arg, which is a special subroutine
of bash; but not so difficult to understand it. (In the followings, we assume /home/binx/ is
your bin directory at which we have all binaries for ecalj.)

### START gwsc: ITER= 10, MPI size= 24, TARGET= gaas
--- No sigm nor sigm.$TARGET files for starting ---
---- goto sc calculation for given sigma-vxc --- ix=,0
No sigm ---> LDA caculation for eigenfunctions
OK! --> Start echo --- | mpirun -np 24 /home/binx/lmf-MPIK gaas > llmf_lda
OK! --> Start echo 0 | /home/binx/lmfgw gaas > llmfgw00
OK! --> Start echo 1 | /home/binx/qg4gw > lqg4gw
OK! --> Start echo 1 | mpirun -np 24 /home/binx/lmfgw-MPIK gaas > llmfgw01
OK! --> Start echo --- | /home/binx/lmf2gw > llmf2gw
... (preparation stage ends here; start main stage) ...
OK! --> Start echo 0 | /home/binx/rdata4gw_v2 > lrdata4gw_v2
OK! --> Start echo 1 | /home/binx/heftet > leftet
OK! --> Start echo 1 | /home/binx/hchknw > lchknw
OK! --> Start echo 3 | /home/binx/hbasfp0 > lbasC
OK! --> Start echo 3 | mpirun -np 24 /home/binx/hvccfp0 > lvccC
OK! --> Start echo 3 | mpirun -np 24 /home/binx/hsfp0_sc > lsxC
OK! --> Start echo 0 | /home/binx/hbasfp0 > lbas
OK! --> Start echo 0 | mpirun -np 24 /home/binx/hvccfp0 > lvcc
OK! --> Start echo 1 | mpirun -np 24 /home/binx/hsfp0_sc > lsx
OK! --> Start echo 11 | mpirun -np 24 /home/binx/hx0fp0_sc > lx0
OK! --> Start echo 2 | mpirun -np 24 /home/binx/hsfp0_sc > lsc
OK! --> Start echo 0 | /home/binx/hqpe_sc > lqpe
OK! --> Start echo --- | mpirun -np 24 /home/binx/lmf-MPIK gaas > llmf_oneshot
... (this is the end of main stage) ...
== 0 iteration over ==
---- goto sc calculation for given sigma-vxc --- ix=,1
OK! --> Start echo --- | mpirun -np 24 /home/binx/lmf-MPIK gaas > llmf
OK! --> Start echo 0 | /home/binx/lmfgw gaas > llmfgw00
... (lines here omitted) ...
OK! --> Start echo 0 | /home/binx/hqpe_sc > lqpe
== 1 iteration over ==
... (lines here omitted) ...
== 2 iteration over ==
... (this repeat until ITER= 10(the first argument to gwsc) ) ...
```


This shows that **gwsc** invoke **lmf-MPIK**, **lmfgw**, **qg4gw**,... successively. **echo 3|hbasfp0** means running a fortran program **hbasfp0** with the argument '3' from the standard input (**read(*,*)** in fortran code). We can divide these successive calls to “preparation stage” and “main stage”. Preparation stage is just to prepare eigenfunctions and so on which are required for the “main stage” of GW calculation. At the end of “main stage”, we have the potential file **sigm**.

As it shows, console output are going to **l*** files.

5.2 Preparation stage of gwsc

At the end of this stage, we get required eigenfunctions, BZmesh data, and so on, which are required for the main stage. Note that **echo 0 |lmfgw** means supply an integer to the fortran program **lmfgw** from standard input by **read(*,*)**.

- **lmf-MPIK** (k-parallel version of **lmf**)
This is the one-body calculation for given **sigm.gaas**. At the beginning, we do not have **sigm.gaas**. In this case **lmf-MPIK** just perform LDA/GGA calculation.
- **echo 0 |lmfgw**:
Get some small information files to start **qg4gw**. If you type just **lmfgw**, and observe what occurs. It shows a menu and pauses (asking you to supply an integer); then we supply 0 in this case. (If we do **echo 0 |lmfgw**, no pause occurs.)
- **echo 1 |qg4gw** : Get **k** points used in the *GW* calculations and the corresponding **G** vectors. And what is the irreducible **k** point (See console output of **qg4gw**. **gwsc** keeps it in **lqg4gw**).
Since we use the offset-Gamma method for BZ integration for $G \times W$, we need shifted mesh points to calculate W at offset-Gamma points. The **q** vectors of offset-Gamma method is in **Q0P** file. (If you have two points in **Q0P**, we see two shifted mesh points in addition to regular mesh points.) Remember that cutoff of **G** is given by **QpGcut_psi** and **QpGcut_cou** in **GWinput**. (Based on the experiences, we use smaller **QpGcut_cou** to reduce computational time. Explained in other section xxx).
- **echo 1 |lmfgw-MPIK** : Calculate the LDA eigenfunctions, eigenvalues, and $\langle \psi | V_{xc}^{LDA} | \psi \rangle$ at the irreducible **k** points (shown at the bottom of output **lqg4gw** of **qg4gw**.)
- **lmf2gw**: store these data into **DATA4GW_V2** and **CphiGeig**, whose I/O is controlled by a key subroutine **gwinput_v2.f**.

5.3 Main stage of gwsc

We can start the main stage of *GW* calculation from these files;

GWinput **DATA4GW_V2** **CphiGeig** **QGpsi** **QGcou** **Q0P** **QIBZ** **SYMOPS** **BZDATA** **HAMindex** **CLASS**;
these files contains eigenfunctions and so on in the manner of Eq.(17) of [Kotani(2014)], eigenvalues and other required information. This is the starting point of the *GW* calculation.

- **GWinput** : computational conditions.
- **DATA4GW_V2** : Crystal structures and so.
- **CphiGeig** : Eigenvalues and Eigenfunctions
- **QGpsi** : **q** and **G** vector for the eigenfunctions(**q** means **k** in the previous section),
- **QGcou** : **q** and **G** vector for the Coulomb matrix
- **Q0P** : **q** points near **q=0** instead of **q=0** (offset Gamma points)
- **QIBZ** : irreducible **q** points (This is also contained in **BZDATA**).
- **SYMOPS** : point group operation
- **BZDATA** : **q** points data (and tetrahedron weights if necessary) for BZ integrals.
- **HAMindex** : Hamiltonian index, all required complex index for Hamiltonian of PMT method. (See the top of subroutine **write_hamindex** in **lm7K/subs/m_hamindex.F**. This is also in

fp_{gw}/gws_{src}/m_{ham}index.F. Identical files are in two different directory— it should be avoided in future.)

- **CLASS** : class information or atomic sites (equivalent sites).

With these files, we do the main stage as

- **rdata4gw_V2**: Read DATA4GW_V2 and so on, and decompose it into files required in the followings. And calculate PPOVL* files (overlap matrix of interstitial plane waves. Because of technical reasons some different types of PPOVL* with **q**-point index).
- **hefttet** : Get the Fermi energy **EFERMI** by tetrahedron method. It is used in **hx0fp0**.
- **hchk_{nw}** : stores the number of required ω points along real-axis into **NW**. (NW is not essentially used, but is supposed to exist in the followings.)
- **echo 3|hbasfp0**: gives the product basis for Core exchange.
- **echo 0|hvccfp0**: gives the Coulomb matrix for the Core exchange.
- **echo 3|hsfp0_sc** : gives the Core exchange part of the self-energy.¹²
- **echo 0|hbasfp0**: gives the product basis.
- **echo 0|hvccfp0**: gives the Coulomb matrix v .
- **echo 1|hsfp0_sc** : gives the exchange part of the self-energy.
- **echo 11|hx0fp0_sc** : gives the correlated part of the screened Coulomb interaction $W - v$.
- **echo 2|hsfp0_sc** : gives the correlated part of the self-energy.
- **echo 0|hqpe_sc**: gather data and write down final results into **sigm**, **QPU**, **TOTE.UP** files.

5.4 Other functions (or scripts)

In addition to **gw_lm_{fh}**, there are some other additional scripts and functions.

- **gw_lm_{fh}** : The one-shot *GW* calculation. Lifetime(impact ionization rate) of QPs.
- **gwsc** : QSGW calculation explained here
- **epsPP_lm_{fh}**, **eps_lm_{fh}** : Dielectric function without or with local-field effects.
- run-mode 4 of **hsfp0**: to plot the spectrum function $\Sigma(\omega)$.(need to be fixed again probably).
- **epsPP_lm_{fh}_chipm** : non-interacting spin susceptibility. One-degree of freedom like Rigid moment approx. After it ends, you need to do **calj_nlfc_{metal}** and/or **calj_summary_{mat}** to get the full spin susceptibility.
- **genMLWF** : Wannier function and its matrix elements of the Screened Coulomb interaction.

¹²Correlation part due to cores is neglected. In future, we will switch to a version without PB for core part to reduce computational time and for numerical accuracy.

6 Cautions for usage

1. == not meaningful total energy in QSGW===
Total energy shown in QSGW mode in current version is not meaningful. (just treat as an indicator to convergence).
2. == Do we use VWN or GGA for QSGW? ===
In principle, QSGW results should not depends on VWN or GGA (XCFUN=1 or 103 in ctrl). But there is minor dependence, because
 1. frozen core density.
 2. core eigenfunctions.
 3. radial basis functions
 4. Slight numerical reason
 (This is probably because Sigma-interpolation procedure But not exactly figured out yet → affect about 0.02eV as for band gap for GaAs.). In anyway, use VWN (HAM_XCFUN=1) as standard. And such technical things affects, 0.05 eV level of error for band gap.

3. EH and EH2 : For si, if EH and EH2 are the same, the following error occurred.

```
fexit,fexit2,fexit3 error retval=          -1
Exit -1 zhev_tk2: nev /=nevx something wrong.
```

The large EH, EH2 get to be meaningless. We usually use up to ~ 2 . (If you use very large EH such as $E \sim 10$, I am not so sure weather it is)

4. The options about the product basis within MT. (SeungWoo's memo)

```
<PRODUCT_BASIS>
tolerance to remove products due to poor linear-independency
0.100000D-02 ! =tolopt
```

When the product basis are made, we may have poorly linear independent basis. For example, one in the set $\{f_1, f_2, \dots, f_n\}$ would be almost give by a linear-combination of others. We need to make the linear-independent set. Therefore, after calculating the overlap matrix $\langle f_i | f_j \rangle$. We do diagonalization, then we remove eigenvectors corresponding to small eigenvalues than **tolopt**. See the **hbasfp0** command in **gwsc**

```
lcutmx(atom) = maximum l-cutoff for the product basis.
4 4 4 2 2 4 4
```

For $\phi_1 \times \phi_2$ case, $|l_1 - l_2| \leq l_{tot} \leq |l_1 + l_2|$. So 'lcutmx' changes the maximum cutoff for the l_{tot} . The order is the same as the order of atoms in the **ctrl** file.

atom	l	nnvv	nnc !
1	0	3	3
1	1	3	2
1	2	2	1

'atom' means the atom number identified in the **ctrl** file.

'l' is the angular momentum quantum number.

'nnvv' is the number of radial functions (valence) on the augmentation-waves.

'nnc' is the number of radial functions for core.

The latter two ones, 'nnvv' and 'nnc', will be understood more clearly if you see the following ones.

atom	l	n	occ	unocc	! Valence(1=yes,0=no)
1	0	1	1	1	! 5S_p -----
1	0	2	0	0	! 5S_d
1	0	3	1	1	! 4S_l
1	1	1	1	1	! 5p_p
1	1	2	0	0	! 5p_d
1	1	3	1	1	! 4p_l

Above options are about the product basis set within MT (Valence).

‘atom’ and ‘l’ are explained above. ‘nnvv’ for ‘atom = 1 and l = 0’ was ‘3’ so this case we have 3 basis (‘n = 1, 2, 3’)

‘n’ is the degree of freedom of the radial function, ϕ . ‘n = 1’ means ϕ , ‘n = 2’ means $\dot{\phi}$, and ‘n = 3’ means kind of $\ddot{\phi}$, which the dot above the letter represents the differentiation with respect to the energy. So ‘n = 1 and 2’ is related to the linearization of the radial function and ‘n = 3’ is the local orbital which is restricted in MT. The local orbital can be modified changing ‘PZ’ in the **ctrl** file. Finally, the number of the basis set which is needed for expanding eigenfunctions is $(l + 1)^2 \times n$.

‘occ’ and ‘unocc’ mean that we use only ones that checked as ‘1’, in other words we neglects ‘0’ cases for making product basis. Be careful for confusion with name ‘occ’ and ‘unocc’. These don’t mean that occupied or unocc. When making product basis, $M = \phi_1 \times \phi_2$, ‘occ’ corresponds to ϕ_1 and ‘unocc’ to ϕ_2 . For example,

atom	l	n	occ	unocc	! Valence(1=yes,0=no)
1	0	1	1	1	! 5S_p -----
2	3	1	0	1	! 4f_p

If the options are like the above, the product basis will be consists of $(\phi_1 = \phi_{atom=1,l=0}) \times (\phi_2 = \phi_{atom=1,l=0})$, $(\phi_{atom=1,l=0} \times \phi_{atom=2,l=3})$. As you can see, $(\phi_1 = \phi_{atom=2,l=3})$ is skipped.

In the **ctrl** file, ‘EH’ controls the l part. As for ‘EH’, (s, p, d, f) are used but **GWinput** file uses (s, p, d, f, g). ‘EH’ : HEAD part. ‘GWinput’ : contains TAIL part... need more explanation.

atom	l	n	occ	unocc	ForX0	ForSxc	! Core (1=yes, 0=no)
1	0	1	0	0	0	0	! 1S -----
1	0	2	0	0	0	0	! 2S
1	0	3	0	0	0	0	! 3S

Above options are about the product basis set within MT (Core).

‘nnc’ for ‘atom = 1 and l = 0’ was ‘3’ so this case we have 3 basis (‘n = 1, 2, 3’)

Finally, for the convergence check, we can modify the following three things, (i) tolerance, (ii) lcutm, and (iii) occ and unocc.

== one show QSGW (not one-shot GW) ==

one-shot QSGW can be useful in cases.

As it contains off-diagonal part, we can resolve band tanglement problem in Ge (no band gap).

== Restart calculation in lda ==

lmf(lmf-MPIK) read rst.* in default.
 rst contains electron density.
 If rst is already converged, it stops after two iteration.
 rst contains atomic positions.
 So, in order to read atomic positions change in ctrl,
 Use options shown in lmf --help.

== Restart calculation in qsgw ==

To remove mixsigm* (mixing for sigm), maybe required.

== iteration check ==

First, watch console output of gwsc (do redirect to output file)

Need to check OK! signs arrayed on 1st columns.

gwsc iteration is time consuming,

So we need to check calculations are normally going on or not.

Memory inefficiency.

Set 'KeepEigen off' and 'KeepPPOVL' off.

In fact, our code is still inefficient for memory usage.

grep gap llmf ---> minimum gap at mesh point.

see save.* ,or grep '[xc]' save.*

the end of iteration of lmf is shown as x or c.

(if failed, QPU file.

dqpu QPU.4run QPU.3run

As for usual semi-conductor, accuracy about 0.1 eV is limit of current implementation.

Set vwn (xcfun=1) looks better (stable) for GW.

\$grep rms lqpe*

shows

```
... rmsdel=2.44D-04
... rmsdel=4.91D-03
... rmsdel=2.44D-04
... rmsdel=3.37D-04
```

If rmsdel is getting to be smaller, it is on convergence path.

(but in magnetic cases, it may give be too good even not yet going to
 be converged..., because magnetic energy is so small)

grep diffe llmf ---> difference of energies of each iteration.

ehf (harris energy)

ehk (Hohenberg kohn energy)

== emax cutoff for APWs. ==

We can not use so many APWs in current version,

because of overcompleteness (this is because null vector within MTs),

In anyway, use pwemax=3 as standard (test it with 4 or 5).

To avoid failure of calculation, we may use smaller MT radius for alkali, and alkali-earth elements.
In future, I think we can introduce pseudopotentials for these atoms only.

== Check Used MTO

Near begining of console output, what MTO you use is shown as: (GaAs case).
sugcut: make orbital-dependent reciprocal vector cutoffs for tol= 1.00E-06

spec	l	rsm	eh	gmax	last term	cutoff
Ga	0*	1.13	-1.00	6.579	1.19E-06	1459
Ga	1*	1.13	-1.00	7.028	1.26E-06	1807
Ga	2*	1.13	-1.00	7.475	1.09E-06	2109
Ga	3	1.13	-1.00	7.920	1.06E-06	2637
Ga	0*	1.13	-2.00	6.579	1.19E-06	1459
Ga	1*	1.13	-2.00	7.028	1.26E-06	1807
Ga	2	1.13	-2.00	7.475	1.09E-06	2109
As	0*	1.18	-1.00	6.300	2.13E-06	1243
As	1*	1.18	-1.00	6.720	1.26E-06	1471
As	2*	1.18	-1.00	7.140	1.37E-06	1837
As	3	1.18	-1.00	7.558	1.05E-06	2229
As	0*	1.18	-2.00	6.300	2.13E-06	1243
As	1*	1.18	-2.00	6.720	1.26E-06	1471
As	2	1.18	-2.00	7.140	1.37E-06	1837

== gwsc cause error stop.

Have you ever changed MTO setting? Consistent with GWinput?

== QSGW for Fe.

It is better to use 3p as core. Furthermore, 3d+4d as valence is better.
Thus we need to set PZ=0,3.9,4.5
I also got aware that emax_sigm should be large enough ($4\epsilon_{\text{sim}}5$ Ry)
to have smooth band dispersion. n1n2n3 can be 10x10x10.

== RSRNGE: enlarge RSRNGE ==

Use RSRNGE=10 or so (in cases, RARNGE=20 or more is required),
for large number of k points. Try and enlarge it if it fails with a
message "Exit -1 rdsigm: Bloch sum deviates more than allowed tolerance (tol=5e-6)".
We will have to make it automatic in future.

== QOP check

In cases, it is better to use QOPchoice=2 instead of default QOPchoice=1.
(For slabs, QOPchoice=2 may be better; need check more. In anyway,
it is problematic to use unbalanced k points for anisotropic cell).
See Copmuter Physics Comm. 176(2007)1-13).

=== When calculation in LDA level fails ===

when calculation fails in LDA level.

- (1) smaller MT
- (2) fewer PW. smaller pwemax.

(3) core as semicore.

=== LDA+U ===

not yet written...

=== MAE by rotating crystal ===

(we have a sample at 1m7K/TESTsmamples/MAEtest/, but only in GGA/LDA).

=== spin wave ===

J calculation.

====

If not stable convergence in gwsc, try to set

mixbeta 0.5

(and/or mixpriorit 3 or something)

at the begining of sigma.

=====

cleargw (directory):

This command clean up up intermediate files under (directory).

This recursively into deeper level. Be careful, or edit it.

I use it as '>cleargw .'.

Magnetic moment within MTs are shown as

charges:		old	new	
smooth		17.240314	17.240740	...
mmom		0.000024	-0.000010	
site	1	6.207135	6.206590	
mmom		1.062276	1.062991	<--- here
site	2	6.207115	6.206834	
mmom		-1.062323	-1.062958	<--- here
site	3	1.172718	1.172918	
mmom		0.000011	-0.000011	
site	4	1.172718	1.172918	
mmom		0.000011	-0.000011	

In this case, MTsite1 has 1.062991 and MTsite2 has -1.062958.

>grep 'lin mix' -A30 llmf

can take out this message (if console output is in llmf).

ORBITAL MOMENT in perturbation:

Try

```
>lmf nio --rs=1,0 -vso=1 --quit=band >llmf
```

After converged, try

```
>grep IORBTM -A20 llmf
```

Then llmf shows orbital moment in first order perturbation.

(Here --rs=1,0 read rst.* file but not change it. See >lmf --help.

--quit=band means quit just after band calculation.)

== EPS mode,

Check Im part of chi0 is smoothly damping at high energy (typically 1Ry or larger energy range). If there is some large Im part remains, something strange (usually due to orthogonality problem of eigenfunctions when you set low q).

Related source codes are in ecalj/lm7K/ .

A command ecalj/lm7K/ctrlgenM1.py can generate 'standard input file (ctrl file)' just from a given crystal structure file called as ctrls file.

Binaries are lmf and lmf-MPIK (MPI k-parallel version).

6.1 lmf -help

lmf -help show option of -rs=(five numbers); this let lmf know how to read atm.* file which is the initial atom file by lmfa.

7 Wannier function

xxxxxxxxxx under construction xxxxxxxxxxxxxxxxxxxxx (See also ecalj/fpgw/Wannier/README)...

We can generate Wannier functions (maximally localized Wannier Functions or similar) by a script **genMLWF**. Run examples in ecalj/fpgw/MATERIALS/*MLWF. To run the script, we need to set options in GWinput. For initial condition, we need

<MLWF>

```
5          # gaussian, nwf
```

```
1 1 1 1 1 # nphi(1:nwf)
```

```
9 14 2.0 1.0 phi,phidot and lamda(angs) of gaussian 1, #iphi(j,i),iphidot(j,i),r0g(j,i),wphi(j,i)
```

```
10 15 2.0 1.0 phi,phidot and lamda(angs) of gaussian 2
```

```
11 16 2.0 1.0 phi,phidot and lamda(angs) of gaussian 3
```

```
12 17 2.0 1.0 phi,phidot and lamda(angs) of gaussian 3
```

```
13 18 2.0 1.0 phi,phidot and lamda(angs) of gaussian 3
```

</MLWF>

In addition, we have some settings (energy windows and so on). This is the example of the initial conditions for Cu case. 5 is the number of Wannier function. The most left one means ϕ index and the right one of it is $\dot{\phi}$ index. They are written in the @MNLA.CPHI file.

Then we can run **genMLWF**. After it finished, we can analyze its results. (if you don't need Wannier function plot, You can skip a line of wanplot in genMLWF. Then we don't need to set vis_wan_* options.)

7.1 lwmatK1 and lwmatK2

If you input the following command

```
>grep Wan lwmatK*
```

You will get the following results. (This case : Cu cases)

```
lwmatK1: Wannier    1    1    24.644475    0.000000 eV
lwmatK1: Wannier    1    2    24.644576    0.000000 eV
lwmatK1: Wannier    1    3    25.471361    0.000000 eV
lwmatK1: Wannier    1    4    24.644575    0.000000 eV
lwmatK1: Wannier    1    5    25.470946    0.000000 eV
lwmatK2: Wannier    1    1    0.000000 eV   -21.263759   -0.000000 eV
lwmatK2: Wannier    1    2    0.000000 eV   -21.263839    0.000000 eV
lwmatK2: Wannier    1    3    0.000000 eV   -21.931033   -0.000000 eV
lwmatK2: Wannier    1    4    0.000000 eV   -21.263839   -0.000000 eV
lwmatK2: Wannier    1    5    0.000000 eV   -21.930702   -0.000000 eV
```

8 ctrl file details

A ctrl file is usually generated from a ctrls file by the `ctrlgenM1.py` (a crystal structure file is not “ctrl” but “ctrls”). It contains self explanation. Here we give complementary explanations to it. Let us Look into a ctrl file. This is a head part of `ctrl1.cu` generated by `ctrlgenM1.py`:

```
### This is generated by ctrlgenM1.py from ctrls
### For tokens, See http://titus.phy.qub.ac.uk/packages/LMT0/tokens.html.
### Do lmf --input to see all effective category and token ###
### It will be not so difficult to edit ctrlge.py for your purpose ###
VERS    LM=7 FP=7          # version check. Fixed.
IO      SHOW=T VERBOS=35 TIM=2,2
        # SHOW=T shows readin data (and default setting at the begining of
        console output)
        # It is useful to check ctrl is read in correctly or not
        (equivalent with --show option).
        # larger VERBOSE gives more detailed console output.
SYMGRP find # 'find' evaluate space-group symmetry automatically.
        # Usually 'find is OK', but lmf may use lower symmetry
...
```

Note that `#` means comment lines. We can also use lines starting from `% const ...` to define variables and set constant.

We see “categories” such as `VERS`, `IO`, and so on. The beginning of categories are starting from the first column. Under categories, we have “tokens” such as `VERBOSE`. Thus we specify full name of token `VERBOSE` under category `IO` as `IO_VERBOSE`.

- `IO_TIM` is for debugging. It shows which subroutines are called and so on. Bigger number shows deeper subroutines.
- `SYMGRP` is a category without token under it; we set generators of space group (See explanation in previous paragraph). When we set `find`, it automatically calculate symmetry of crystal lattice. If we like to enforce symmetry, set some of generators which are shown by `lmchk`.

- We see `ctrls` is embedded in the `ctrl` by `ctrlgenM1.py`.

```
... (skip) ...
% 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
        NL=4 NBAS= 1 NSPEC=1
SITE    ATOM=Cu POS=0 0 0
... (skip) ...
```

NL, NBAS(number of SITE) and NSPEC(number of SPEC) are automatically added by `ctrlgenM1.py`. It is possible to deform unit cell by adding optional tokens (it is possible to rotate PLAT for magnetic anisotropy calculation). See <http://titus.phy.qub.ac.uk/packages/LMTO/tokens.html#>. For new calculations, it is better to find some examples first.

- **SITE** category: As for MT sites, we have two categories. (1)SPEC(species) and (2)SITE(specify centers of atoms(species) in primitive cell). As for SPEC, we specify MTs(radius, Z, MTOs on it) appeared in the cell. These are defined subtokens under `SPEC_ATOM=foobar` (we have multiple `SPEC_ATOM=foobar`).

Then we place these MTs at SITE sections in the cell. At SITE, we specify atomic sites (What `SPEC_ATOM` is placed to positions by POS) in a primitive cell. We set `POS=` by direct form (Cartesian) but with the unit of `ALAT+DLAT`. Total number of SITE (number of tokens `SPEC_ATOM`) is the number of atoms in the primitive cell. Setting `POS=` under `SPEC_ATOM=foobar` means that we place MT named as `foobar` defined in `SPEC_ATOM=foobar`. In addition, we can set `SPEC_ATOM.RELAX`, if you like to find relaxed structure (we simultaneously set DYN category) in LDA. As for relaxation, see `LaGa0_relax/ctrl.lagao` example, and read <http://titus.phy.qub.ac.uk/packages/LMTO/tokens.html#DYNcat>.

The `SPEC_ATOM=foobar` (with same `foobar` with different POS) are not necessarily equivalent with respect to the space group operation of a system. Thus `SPEC_ATOM=foobar` are divided into “classes” which are connected by the operation. The `lmf` automatically judge “classes” (see also info by `lmchk`). Thus not need to specify it, but it may be better to check it. A sample is `lmchk lagao` at `~/ecalj/lm7K/TESTsamples/LaGa0_relax`

- **SPEC** category: In `ctrls`, we have not yet specified contents of SPEC; we have just given default symbols or only Z= when we use non-default names (shown by `ctrlgenM1.py -showatomlist`). The command `ctrlgenM1.py` adds default SPEC sections.

We have some `SPEC_ATOM`, under which we give subtokens such as `SPEC_ATOM_R`(MT radius), `SPEC_ATOM_Z`(nucleus charge), cutoff parameters of angular momentum, and so on. These `SPEC_ATOM` is refereed to in SITE.

An example of SPEC category is

```
SPEC
ATOM=Fe Z=26 R=1.70
KMXA={kma} LMX=3 LMXA=4 NMCORE=1
PZ=0,3.9,4.5
EH=-1 -1 -1 -1 RSMH=0.85 0.85 0.85 0.85
EH2=-2 -2 -2 RSMH2=0.85 0.85 0.85
MMOM=0 0 2 0
```

ATOM=... (then the similar block of ATOM= are repeated.)

...

Under the token `ATOM=Fe`, we have subtokens `SPEC_ATOM_Z`, `SPEC_ATOM_R`, and so on.

Subtokens `Z` is the nucleus charge and `R` = MT radius. Note that `Fe` is just a name to distinguish MT sphere in the cell. If you set `SPEC_ATOM_Z=27`, it is recognized as `Co` (since `Z=27`). `LMX=3` is the maximum `l` of MTOs. Thus maximum `l` of MTO is `l=3`. The maximum of `l` to expand electron density and potential within MT is `LMXA` (in contrast to usual LAPW), we can use quite small `LMXA` such as `LMXA=4`. `NMCORE=1` means we calculate core density without non magnetically-polarization. This can reduce computational confusion.

`PZ` is to set local orbital (if not, no local orbitals). `EH` and `RSMH` are to specify first set of MTOs. (We can check how local orbitals are set by `lmfa` explained in the next section). `EH2` and `RSMH2` are to specify second set of MTOs.

After `PZ=`, we have three numbers. These are numbers for `s,p,d,f,g,...` channels. Zero means not exist. You can use space or comma(,) as delimiter. Here not only the integer part of principle quantum number, but also the fractional part should be supplied (If `PZ=0,3,4`, it does not work.) Now `PZ=3.9` for `p` and `PZ=4.5` for `d`. This means we use local orbital for `3p`, and local orbital for `4d` (fractional parts (continuous principle quantum number) are large ~ 0.9 for core like orbital, and smaller for extended orbital ~ 0.3 or something. See Logarithmic Derivative Parameters at <http://titus.phy.qub.ac.uk/packages/LMTO/lmto.html>). This is a little confusing, thus we will explain this in appendix. See Sec.??.

`EH`(damping factor), and `RSMH` (where the smooth Hankel function bent) determines MTOs (or its envelope function as a smooth Hankel function). We now set four numbers for them. Thus we set MTOs `s,p,d,f` with `EH=-1` and `RSMH=0.85`. Our current test shows that `RSMH` is one half of `R` (that is, $0.85=1.70/2$, but minimum `RSMH` is 0.5) and not need to be dependent on `s,p,d,f`. (If `LMX=2`, `s,p,d` are allowed and no `f` MTOs.) `EH` is -1; not need to change except test purpose. In a similar manner, `EH2` and `RSMH2` for second set of MTOs are given. Just three numbers means these for `s,p,d`.

`MMOM=s,p,d,f...` gives initial condition of magnetic moment in μ_B (number of up-down electron).

In cases such as `As`, the local orbital given by default `ctrl` is responsible of rather deep core, and it is not need to be treated as valence electrons. In such a case, we don't need local orbital.

In the case of AntiFerro-II `NiO`, it contains two `NiO` in a primitive cell. Thus it is reasonable to have two `SPEC_ATOM` as `Ni1` and `Ni2`, although subtokens under `ATOM=Ni1` and `ATOM=Ni2` (e.g. `SPEC_ATOM_EH` for them) are the same except initial condition of magnetic moment of `MMOM=s,p,d,f...` See example of `NiO`.

The minimum help of call `Category_token_subtoken` are listed with minimum explanation with

```
$ lmf --input
```

It gives a long output. But many of them are experimental and not need to manage them. A part of it is

```
Token          Input    cast  (size,min)
-----
... ..

STRUC_ALAT      reqd    r8      1, 1
Scaling of lattice vectors, in a.u.
... ..
```

This is an minimum explanation of it. "reqd" means "required" (no default). r8 means it read with real number, 1,1 means that ALAT=xxx should contain one number minimum (max is also one) (See also STRUC_PLAT, and so on).

There are kinds of examples in ecalj packages. Please look into their ctrl.* and ctrl.* These are in lm7K/TESTsample/* and ecalj/CMDsamples. In addition, ecalj/MATERIALS contain many samples (need a command); see a later subsection.

As for what is shown in \$ lmf --input, most of important tokens are already described in the ctrl file generated by ctrlgenM1.py. So, we don't need to care many options shown by it.

But we have not yet explained some useful features; STRUC category to deform crystal; DYN category for dynamics; LDA+U treatment; Adding background charge; Core-Hole treatments. We will prepare examples for it if requested.

<http://titus.phy.qub.ac.uk/packages/LMT0/tokens.html#STRUCcat>

<http://titus.phy.qub.ac.uk/packages/LMT0/tokens.html#DYNcat>

For QSGW calculation:

We need a setting in ctrl file to read sigm file (HAM_SIGP). It is simplified now, and not need to care it so much. As we set RDSIG=12 in defaults, lmf read sigm file and add it to one-body potential as long as sigm.* exist.

NOTE for old users: We now set SIGP[MODE=3 EMAX=9999.] in ctrl file to read self-energy in lmf (or lmf-MPIK). This is because we use very localized MTOs (similar with the Maxloc Wannier). Our test shows reasonable results and this simplify algorithms. In my previous version, we asked you to use SIGP[MODE=3 EMAX=2.0] where EMAX is a little (0.5Ry) less than `emax_sigm`. If something strange occurs, try this setting).

- In principle, QSGW result should not depended on the choice of XCFUN. However, it can affect slightly. In our tests, it seems slightly better to use VWN (XCFUN=1) for QSGW calculations. (BUT need to check more...)

8.1 How to set local orbitals

As we stated, do "lmfa |grep conf" to check used MTO basis.

We have to set SPEC_ATOM_PZ=?,?,?

(they ordered as PZ=s,p,d,f,...) to set local orbitals.

lmv7 (originally due to ASA in Stuttgart) uses a special terminology "continous principle quantum number for each l", which is just related to the logalismic derivative of radial funcitons at MT boundary. It is defined as

$P = \text{principleQuantumNumber} + 0.5 - 1/\pi * \text{atan}(r * 1/\phi \, d\phi/dr)$,
 where ϕ is the radial function for each l. For example,
 $P = n.5$ for $l=0$ of free electron (flat potential) because $\phi=r^0$,
 $P = n.25$ for $l=1$ because $\phi=r^1$;
 $P = n.147584$ for $l=2$ because $\phi=r^2$; $P = n.102416$ $n.077979$ for $l=3,4$.
 (Integer part can be changed). See Logarithmic Derivative Parameters in
<http://titus.phy.qub.ac.uk/packages/LMTO/lmto.html#section2>

Its fractioanl part $0.5 - \text{atan}(1/\phi \, d\phi/dr)$ is closer to unity for core like orbital, but closer to zero for extended orbitals.

Examples of choice:

Ga p: in this case, choice 0 or choice 2 is recommended.

We usually use lo for semi-core, or virtually unoccupied level.

(0)no lo (4p as valence is default treatment without lo.)

3p core, 4p valence, no lo: default.

Then we have choice that lo is set to be for 3p,4p,5p.

(1)3p lo ---> 4p val (when 3p is treated as valence)

3d semi core, 4d valence

Set PZ=0,3.9

(P is not requied to set. *.9 for core like state. It is just an initial condition.)

(2)5p lo ---> 4p val (PZ>P)

Set PZ=0,5.5

5.5 is just simply given by a guess (no method have yet

implemented for

If 5.2 or something, it may fails

because of poorness in linear-dependency. We may need to observe
 results should not change so much on the value of PZ.

(3xxx)4p lo ---> 5p val (we don't use this usually. this is for test purpose)

4p lo, 5p valence

Set PZ=0,4.5 P=0,5.5 (In this case, set P= simultaneously).

(NOTE: zero for s channel is to use defalut numbers for s)

Ga d: (in this case, choice 0 or choice 1 is recommended).

(0)no lo (3d core, 4d valecne, no lo: default.)

Then we have choice that lo is set to be for 3d,4d,5d.

(1) 3d lo ---> 4d val (when 3d is treated as valence)

Set PZ=0,0,3.9 (P is not required to set)

(2) 5d lo ---> 4d val (PZ>P)

Set PZ=0,0,5.5

(3xxx) 4d lo ---> 5d val (this is for test purpose)

Set PZ=0,0,5.5 P=0,0,4.5

(NOTE: zero for s,p are to use default numbers)

If you like to read from atm.ga file instead of rst file(if exist).

You have to do lmf --rs=1,1,0,0,1, for example. See lmf --help

Because rst file keeps the setting of MT0, thus change in ctrl is not reflected without the above option to lmf.

=====

9 GWinput details

9.1 generate a template of GWinput

As in the previous section, we need two input files `ctrl.si` and `GWinput`. In principle, these two determines final results uniquely. A template `GWinput.tmp` is generated by `mkGWIN_lmf2`. Required files are

Input files

- `ctrl.si` : The control file for PMT method.

(Recently modified `mkGWIN_lmf2` runs `lmfa` internally. If you use older version, do `lmfa` in advance).

Output files

- `GWinput.tmp` : A file including computational conditions for the *GW* calculation. In addition, it specifies the **k** points for which you calculate the QP energy.

When you run `mkGWIN_lmf`, it asks you to supply three numbers for BZ integration as
== Type three integers n1 n2 n3 for Brillouin Zone meshing for GW! ==
n1=

Then you need to type a number e.g. as "2" for n1. Then you need to repeat it for n2 and n3 as

n1= 2

n2= 2

n3= 2

. These numbers specifies what **k** points in BZ is used for BZ integration (In this case, $2 \times 2 \times 2 = 8$ **k** point in the 1st BZ is used. Roughly speaking, we need $4 \times 4 \times 4$ (or $6 \times 6 \times 6$) to get band gap for Si and so on, with ≈ 0.1 eV accuracy.)

Then you have to edit `GWinput.tmp` and copy it to `GWinput`. We details the `GWinput` in later chapter.

We need to repeat `mkGWIN_lmf2` when you change MTO sections in `ctrl` file (adding PZ case, and so on).

9.2 overview of GWinput

(Because of historical reason, input file is different from `ctrl.*`).

The main input files is `GWinput`. This controls the setting of *GW* calculation. The file `GWinput` consists of structures as

keyword1 data1

keyword2 data2

...

In each lines, it consists of keyword and data. Data can be single or plural. As for keywords, upper case or Lowercase is not distinguished. All keywords should start from 1st column (no space at head). Order of lines are irrelevant. As for logical variable, you can use anything "true, yes, on, 1, T" for .true., and anything "false, no, off, 0, F" for .false.

Or we have "tag sections" in `GWinput` specified by `<PRODUCT_BASIS>`, `<QPNT>`, `<PBASMAX>`, `<QforEPS>`, and `<QforEPSL>`. (`<PRODUCT_BASIS>` is requires for all kinds of calculations. `<PBASMAX>` is optional. `<QforEPS>` and/or `<QforEPSL>` are required for epsilon mode). It is like

`<PRODUCT_BASIS>`

tolerance to remove products

```

0.100000D-02 ! =tolopt
lcutmx(atom)
3 3
atom 1
...
</PRODUCT_BASIS>

```

. In these tag sections, you have to keep format for its own (usually numbers are read by free format `read(*,*)`).

The fundamental readin routine for `GWinput` is a subroutine `getkeyvalue` defined in `gwsrc/keyvalue.F`. `getkeyvalue` is a general and convenient readin routine in full use of the f90 features. Read a head part of the file and try to do "grep getkeyvalue *.F" in `gwsrc/` or `main/` so as to see how to use it (test routine is `main/kino_input_test.F`.)

So the `GWinput` consists of three sections

1. General section
2. `<PRODUCT_BASIS>` section
3. `<QforEPS>`, `<QforEPSL>` section (only effective for dielectric function mode)
4. `<QPNT>` section (only effective for one-shot mode)
5. `<PBASMAX>` section (optional)

We will explain each by each in the followings.

9.3 General section

In general section, it looks like

```
! ##### From GWIN0 #####
n1n2n3      1      1      1 ! for BZ meshing in GW
QpGcut_psi   4.000 !(See unit_2pioa for unit) |q+G| cutoff for eigenfunction.
QpGcut_cou   3.000 !(See unit_2pioa for unit) |q+G| cutoff for Coulomb and W.
unit_2pioa off ! off --> a.u.; on--> unit of QpGcut_* are in 2*pi/alat
alpha_OffG   1.000 !(a.u.) Used in auxially function in the offset-Gamma method.
!emax_chi0   99999.000 !(Ry) emax cutoff for chi0 (Optional)
emax_sigma   3.000 !(Ry) emax cutoff for Sigma

! ##### FREQUENCIES from GWIN_V2 #####
dw           0.005000 !(a.u.) energy-mesh (bin width size) along real axis.
omg_c        0.040 !(a.u.) energy-mesh is twiced at omg_c
! coarser mesh for higher energy. Width get to be doubled at omg_c.
iSigMode     3 ! QSGW mode switch for gwsc. use =3.
niw          10 ! Number of frequencies along Im axis. Used for integration to get Sigma_c
! E.g. try niw=6 and niw=12
delta        -0.10D-05 !(a.u.) Broadening of x0. negative means tetrahedron method.
! used by hx0fp0. You get smeard x0 with abs(delta).
deltaw       0.020000 !(a.u.) Mesh for numerical derivative to get the Z factor
esmr         0.003000 !(Ry) used by hsfp0. Keep esmr smaller than band gap for insulators
! Poles of G^LDA are treated as if they have width esmr in hsfp0.
! Change esmr for metals. See DOSACC*---especcially around Ef.
GaussSmear on ! Gaussian or Rectangular smearing for Pole of G^LDA with esmr for hsfp0.
```

1. BZ integration.

`n1n2n3` 3 integers as N_1, N_2, N_3 (no default); They are ≥ 0 .

Brillouin Zone mesh for integration is determined by keywords `BZmesh` and `n1n2n3`. Current version only allow regular mesh point including Gamma point for $G \times W$. But not that `Chi_RegQbz` below allow you to use off-Gamma mesh for $W(\omega)$ (and dielectric function mode).

We usually take '4 4 4', '6 6 6' or '8 8 8' for GaAs. For metal such as Fe, '10 10 10' or more is better.

`Chi_RegQbz` (on or off)

`Chi_RegQbz` = on (default): Use regular mesh (including gamma) for eps calculation.

`Chi_RegQbz` = off : Use off-Gamma mesh (Not including gamma) for eps calculation.

(In cases, `Chi_RegQbz` off gives faster convergence as for `n1n2n3`; not only for GW, but also for dielectric functions `eps_lmfh`.)

2. Plane wave ($\mathbf{q} + \mathbf{G}$) cutoff

`QpGcut_psi` 1 real (no default)

`QpGcut_Cou` 1 real (no default)

`unit_2pioa` 1 logical (no default)

We have two cutoff for $\mathbf{q} + \mathbf{G}$. `QpGcut_psi` is the cutoff of $|q + G|$ for the IPW in the expansion of the eigenfunctions. `QpGcut_Cou` is for the IPW of the interactions v, D, W .

Its unit is specified by `unit_2pioa` ; "off" means unit in a.u. and "on" means in unit of $\frac{2\pi}{\text{alat}}$. (alat is length scale unit in ctrl.*).

Rule of thumb: `QGcut_psi` is a little (usually 0.5 or so) larger than `QpGcut_cou`. It becomes accurate if we use large `QpGcut_cou`. But it enlarge size of IPW(interstitial plane wave) part of Mixed product basis. For test, try 2.7, 3.2, 3.7 for `QGcut_cou` (and add 0.5 or 1 for `QGcut_psi`). Larger one is expensive.

We expand eigenfunctions in the Muffin-tin division of the space. See Eq.[?] in Ref.[?]; in the current *GW* implementation, we use very simple form of eigenfuncitons (not by the 3-component formalism in the [?]).

Thus the form of expansion is just related to the division of space; not directly related to the difference among LAPW, LMTO, and PMT.

3. Eigenfunctions within MTs (no parameters setting for them).

The radial functions (phi and phidot for each l), corresponding to the true parts, (= 2nd component in the 3-component formalism [?]), are automatically determined already in the one-body part of program **lmf-MPIK**.

4. Cutoff for used bands.

`emax_chi0`: 1 real (optional,default= ∞), in Ry

`emax_sigm`: 1 real (optional,default= ∞ ; We usually use 3 Ry).

`emax_sigm` is the maximum limit of the self-energy (measured from the Fermi energy). See the paper [?] which shows how the results are affected by `emax_sigm`. But in cases, small `emax_sigm` can give poor dispersion curve (slightly unnatural behavior) because of sudden cutoff by `emax_sigm`. However, we like to use smaller value to reduce computational time.

That is, larger is better, but expensive (And note that we simultaneously need to use empty spheres when we use large `emax_sigm`, as shown in [?],).

Generally speaking, accuracy less than $\sim 0.1\text{eV}$ (for bandgap) is allowance of current technique. Probably, it may be possible to have better accuracy, but it may ask us to repeat many calculations with changing conditions to confirm stability.

`nband_chi0`: 1 integer (optional,default= ∞)

`nband_sigm`: 1 integer (optional,default= ∞)

These specify how many bands you use in **hx0fp0** (for chi0) and in **hsfp0** (for sigma). Higher bands above them are neglected.

5. Energy mesh related parameters.

`dw`: 1 real (a.u.). Mesh width along real axis for $W(\omega)$.

`omg_c`: 1 real (a.u.).

`dw` and `omg_c` determines ω mesh along real axis to calculate $W(\omega)$. In other words, `dw` and `omg_c` specify real space bins which we accumulate imaginary part weight of polarization functions. `dw` is bin width at $\omega = 0$, then bin width is twiced at `omg_c`. (Energy mesh is getting coarser at higher energy; in other words, the bin width is quadratically larger.) This choice of getting coarser is because we think $W(\omega)$ around $\omega \sim 0$ gives most important contribution to the GW approximation. If bins are too wide, dielectric function can be less accurate, but results are not necessarily so much affected. For metal, our code can capture Drude weight numerically. We do not need to be so sensitive to the choice of them usually.

delta: 1 real (a.u.). We usually use very small number as -1d-8 for gw.lmfh, eps mode and so.

xxx does this really make the stabilization? xxx This is the size of δ in denominator of Π (EQ.xxx). But (I think) we can (or can not) use it so as to make broadening for theoretical test (maybe not exactly corresponding to δ). or when you make calculation stabilized (Takao need to check this point again xxx!)

[Old note. Need check xxx: In gw.lmf, it is used for broadening of x_0 when it call hx0fp0. Then **delta** is δ is EQ.32. The sign of **delta** is just used as a flag whether you use the tetrahedron method of dielectric constant [Rath and Freeman(1975)] or not; minus sign means "Use the tetrahedron method for D "; plus sign means you do it by simple sum. You can usually use this default setting. But it might be possible to use a larger value to smear the fine structures on the energy-dependence of W in cases. This might be necessary if W is so energy-dependent and **dw** is not so small to resolve the structure—but I don't know.]

niw: 1 integer.

Number of integration points along the imaginary axis(FIG.1) to get Σ_c . See routines **wint*** called from **sxcf*.F**, which is called from the main routine **hsfp0.m.F** (or **hsfp0.sc.m.F** in the QSGW case). The integration points are $i\omega'(n) = i(1/x(n) - 1)$, where $x(n)$ is the usual Gaussian-integration points for the interval $[0,1]$. In addition, we give the special analytical treatment for the peaky part at $\omega' = 0$. Out tests shows **niw**=6 for Si is good enough for 0.01 eV accuracy. The convergence as for **niw** is quite good. This integration scheme has been developed by Ferdi Aryasetiawan. The number of points should be the one of 6,10,12,16,20,24,32,40, or 48. It is because we use a subroutine **gauss** in **/gwsr/mate.F** prepared by Ferdi. We will replace better one in future. See II-F in Ref.I.

GaussSmear: 1 logical

esmr: 1 real (Ry). Used by **hsfp0** (and **hsfp0.sc** for QSGW).

Poles of the Green function G^{LDA} are treated as if they have width **esmr** in **hsfp0**. If **GaussSmear** is on, each pole of G^{LDA} is smeared by a Gaussian function with $\sigma = \text{esmr}$ in the calculation of **hsfp0**. If **GaussSmear** is off, we assume rectangular smearing for the poles. Usually it is necessary to take rather smaller value than band gap for insulators. Try to use 0.003 or so in the case of Si and **GaussSmear**=on.

For metal, this **esmr** is somehow related to how we capture the Fermi surface; In principle, we have to take the limit **n1n2n3** $\rightarrow \infty$ and **esmr** $\rightarrow 0$). However, we may inevitably use some finite **esmr** to make calculations converged.)

deltaw: 1 real (a.u.) only for one-shot case.

deltaw is the interval for the numerical derivative $\frac{\partial \Sigma(\omega)}{\partial \omega}$ in EQ.8. We calculate $\langle \psi^{\mathbf{k}n} | \Sigma(\epsilon^{\mathbf{k}n} + \text{deltaw}) | \psi^{\mathbf{k}n} \rangle$ and $\langle \psi^{\mathbf{k}n} | \Sigma(\psi^{\mathbf{k}n} - \text{deltaw}) | \psi^{\mathbf{k}n} \rangle$ in addition to $\langle \psi^{\mathbf{k}n} | \Sigma(\epsilon^{\mathbf{k}n}) | \psi^{\mathbf{k}n} \rangle$. From these values, we can calculate two Z (or second-derivative of $\Sigma(\omega)$), as shown in **SECU**. It will help to see whether the used **deltaw** is O.K. or not.

6. Offset-gamma point.

QOPChoice 0: 1 integer

QOP.Choice gives how to determine the offset gamma points. Initially we take them as

1: —q— is ten times smaller than regular mesh.(default)

2: —q— is average in the Gamma cell (cell of BZ including Gamma point).

Then we choose only inequivalent **q** points based on the point group symmetry. Obtained offset gamma points is given in **QOP** file.

alpha_offG: 1 real (a.u.)

alpha_offG corresponds to α in EQ.48. **alpha_offG=1d0** is usually good in the sense that it seems to be almost a limit at $\alpha \rightarrow 0$. So you can usually fix it as **alpha_offG=1d0**, and check the convergence as for **n1n2n3**.

7. core orthogonalization (default=off)

CoreOrth 1 logical — recently, this option is not maintained — Better to use local orbital instead, so that core charge do not spill out.

If this is on, we enforce cores orthogonalized to valence ϕ and $\dot{\phi}$ (these appear in II-C in Ref.I). This procedure enforce the correct orthogonal condition, thus we have correct behavior for the dielectric function at $\mathbf{q} \rightarrow 0$. However, it may deform core functions too much, especially in the case of shallow 3d (or maybe 4d) cores. So we don't recommend use this option, even though then the orthogonality condition is somehow broken. Anyway you can check weather it affects to results or not by this switch.

8. QP self-consistent GW.

iSigMode 1 integer (no default).

This is required for QSGW calculation by the script **gws**. We have some possible ways to make GW self-consistent (how we determine V_{xc} from calculated $\Sigma(\omega)$). We now mostly use **iSigMode=3**.

3: Use $\text{Re} \frac{\Sigma_{nn'}(\epsilon_n) + \Sigma_{nn'}(\epsilon_{n'})}{2}$ (mode-A in [?]).

1: Use $\Sigma_{nn'}(E_F) + \delta_{nn'}(\Sigma_{nn'}(\epsilon_n) - \Sigma_{nn'}(E_F))$ (mode-B in [?]).

5: Use $\delta_{nn'}\Sigma_{nn}(\epsilon_n)$ (Eigenvalue-only self-consistency, keeping the eigenfuncions as given)).

See **/gwsrsc/sxsf_fa12.sc.F**, which is called from the main routine **hsfp0_sc** (this is the routine to calculate self-energy)).

9. Others

KeepEigen 1 logical (default=on)

These are for memory usage. When **KeepEigen** is on, eigenfunctions (Eigen) are kept in memory during calculation. If you have not enough memory in your machine, use them off. Then you can save memory usage. However, then we may have too frequent access to files. So %CPU might get lower. Be careful to use these options.

Verbose 1 integer (default=0) If 0, it gives minimum standard output. If 40 or higher, it shows too much output. (these verbosity control is not well-organized yet).

10. **LFC@Gamma**, **EIBZmode**, **multitet** are for test purpose.xxx

9.4 <QPNT> section

(only for one-shot GW. Not suitable to make band plot in BZ.) This section is to specify the q points and bands index for which you calculate the QP energies (QPE). An example is

```
<QPNT>
--- Specify the q and band indeces, for which we evaluate the self-energy ---
*** all q -->1, otherwise 0; up only -->1, otherwise 0
      0          0
*** no. states and band index for calculation.
      3
    15 16 17
*** q-points, which shoud be in qbz. See KPNTin1BZ.
      2
```

```

1      0.0000000000000000      0.0000000000000000      0.0000000000000000
2      0.2886751358562925      -0.5000000000000000      0.0000000000000000
3      0.0000000000000000      0.0000000000000000      0.3073140749846343
</QPNT>

```

Numbers are read by free format read(5,*), thus the numbers should be separated by space. At the next line to the first ***, you have to give two numbers used as flags. Both of them takes 0 or 1. 1st one is whether you calculate QPE for all q points (in IBZ) or not. If it is 1, you calculate QPE for all q. If it is 0, you calculate them only for q points specified within this file. In the case of metal where you want to calculate the Fermi energy for QPE, you need to calculate all the eigenvalues somehow above the Fermi energy (If you put 1, it is safer but too time-consuming). The second number is whether you calculate QPE for both spins or not. It is usually 0. In the case of antiferro material, it should be 1.

From the next line to the second ***, you have to specify the states for which you calculate the QPE. In this example, you calculate the 3 bands of QPE for 15th, 16th, and 17th eigenfunctions (they are ordered from the bottom).

From the next line to the third ***, you have to specify the q points. The first numbers of each line are dummy. In this case, you calculate QPE for two q points. The third q point is neglected because 2 is given at first.

When you generate GWinput.tmp, you see all the possible q points are listed (these q points should be a part of the regular mesh points).

In the QSGWmode (gWSC), this section is neglected (then we calculate all QPE on regular mesh points); so its hsfp0_sc part is quite expensive (usually it takes time more than hx0fp0).

Additional Note —————

QPNT_nbandrange num1 num2 (two integers).

This override setting in <QPNT>. (I think this switch may still work, but not checked recently).

AnyQ on (default is off)

If this is on, you can specify any Q point which is not on the mesh point. For the purpose, we need to prepare eigenfunctions at extra \mathbf{k} points. But it is automatic. In order to make the computation efficient. Even in this case, from the computational view, it is better to choose \mathbf{q} on the two times finer divided mesh (or three times finer divided \mathbf{k} mesh). This is used for Fig.6 in Phys. Rev. B 74, 245125 (2006).

9.5 set QPNT for eps mode (QforEPS section)

For eps modes (scripts `eps_*`, which are for linear responses. See Sec.14), you have to specify q point in the following ways.

1. `QforEPSIBZ` on

Then all Q point in IBZ are used.

2. Use section as

```
<QforEPS>
0d0 0d0 0.01d0
0d0 0d0 0.02d0
0d0 0d0 0.04d0
0d0 0d0 0.08d0
</QforEPS>
```

In addition, you can specify Q points as

```
<QforEPSL>
0d0 0d0 0d0 1d0 0d0 0d0 8
0d0 0d0 0d0 .5d0 .5d0 0d0 8
</QforEPSL>
```

This is along the line— 8 point along the line (not left-end q; so omitting 0 0 0). The first line means line (0d0 0d0 0d0)—(1d0 0d0 0d0) is divided to 8. So we have 7 points, (0.125 0 0), (0.25 0 0),... (1 0 0).

9.6 <PRODUCT_BASIS> section

This section is to define product basis to expand W and so. Numbers are read by free format read(5,*), thus the numbers should be separated by space. The line number in this section is meaningful (you can not add comment lines).

```

<PRODUCT_BASIS>
tolerance to remove products due to poor linear-independency
0.100000D-04 ! =tolopt; larger gives smaller num. of product basis. See lbas and lbasC, which are outp
lcutmx(atom) = maximum l-cutoff for the product basis. =4 is required for atoms with valence d, like l
4 3
atom 1 nnvv nnc ! nnvv: num. of radial functions (valence) on the augmentation-waves, nnc: num. f
1 0 2 3
1 1 2 2
1 2 3 0
1 3 2 0
1 4 2 0
2 0 2 1
2 1 2 0
2 2 2 0
2 3 2 0
2 4 2 0
atom 1 n occ unocc ! Valence(1=yes,0=no)
1 0 1 1 1 ! 4S_p -----
1 0 2 1 0 ! 4S_d
1 1 1 1 1 ! 4P_p
1 1 2 0 0 ! 4P_d
1 2 1 1 1 ! 4D_p
1 2 2 0 0 ! 4D_d
1 2 3 1 1 ! 3D_l
1 3 1 0 1 ! 4f_p
1 3 2 0 0 ! 4f_d
1 4 1 0 0 ! 5g_p
1 4 2 0 0 ! 5g_d
2 0 1 1 1 ! 2S_p -----
2 0 2 0 0 ! 2S_d
2 1 1 1 1 ! 2P_p
2 1 2 0 0 ! 2P_d
2 2 1 1 1 ! 3d_p
2 2 2 0 0 ! 3d_d
2 3 1 0 1 ! 4f_p
2 3 2 0 0 ! 4f_d
2 4 1 0 0 ! 5g_p
2 4 2 0 0 ! 5g_d
atom 1 n occ unocc ForX0 ForSxc ! Core (1=yes, 0=no)
1 0 1 0 0 0 0 ! 1S -----
1 0 2 0 0 0 0 ! 2S
1 0 3 0 0 0 0 ! 3S
1 1 1 0 0 0 0 ! 2P
1 1 2 0 0 0 0 ! 3P
2 0 1 0 0 0 0 ! 1S -----
</PRODUCT_BASIS>

```

This section is read in the free format in fortran. So, e.g., 0.01 works as same as 0.10000D-01. The line order is important (you have to keep the order given by `GWinput.tmp`). Be careful atom atom id—lmf may re-order it and pass it to gw code. Look into LMTO file (generated by `mkGWIN_lmf2`); which contains crystal structure information after such re-ordering by lmf. I used ! to make clear that things after ! are comments. But ! is not meaningful – just the expected numbers of data separated by blank(s) are read for each line from the beginning of lines.

- 0.100000D-02 ! =tolopt controls a number of Product basis to expand the Coulomb interaction within MTs. `tolopt` is a criterion to remove the poorly linear-independent product basis. Note that the product basis, which is to expand the Coulomb interaction, is different from the basis to expand eigenfunctions. In our experience, 0.100000D-02 (=0.001) is not so bad. If you like to reduce computational time use 0.01 or so, but a little dangerous in cases. With 0.0001, we can check stability on it.
(note: By supplying multiple numbers, we can specify `tolopt` atom by atom. Remember `lmchk` gives atom ID.)

- `lcutmx(atom)` is the l cutoff of product basis for atoms in the primitive cell (do `lmchk` for atom id). In the case of Oxygen, we can usually use `lcutmx=2` (need check by the difference when you use `lcutmx=2` or `lcutmx=4`). Then the computational time is reduced well.
- (dec2014:<PBASMAX> is not checked recently; see `fpgw/main/hbasfp0.m.F` and `fpgw/gwsrc/basnfp.F`.) You can use <PBASMAX> section to override this setting. It is given as

```
<PBASMAX>
1  5 5 5 3 3
2  5 5 3 2 3
3  3 3 2 2 2
</PBASMAX>
```

The first number is for atom index (fixed), and other are product basis for each l channel.

- The integer numbers in 4th line `lcutmx` gives the maximum angular momentum l for the product basis for each atomic site. In our experience, `lcutmx=4` is required when the semi-core (or valence) $3d$ electrons exist and we want to calculate the QP energies of them.
- Keep a block starting from "atom l nnvv nnc ..." as it originally generated in `GWinput.tmp`. It just shows that how many kinds of radial functions for cores and valence electrons for each atom and l . `nnvv=2` in the case of ϕ and $\dot{\phi}$; `nnvv=3` in the case to add the local orbital in addition.
- There are two blocks after the line "atom l n occ unocc :Valence(1=yes, 0=no)" and after "atom l n occ unocc ForX0 ForSxc ! Core (1=yes, 0=no)". These are used to choose atomic basis to construct the product basis. The product basis are generated from the products of two atomic basis.

`GWinput.tmp` generated by `mkGWIN.lmf2` contains labels on each orbitals as `4S_p`, `4S_d`, `4P_p`... Here `4S_p` is for ϕ_{4s} ; `4S_d` for $\dot{\phi}_{4s}$; `3D_1` for ϕ_{3d}^{local} . Capital letter just after the principle-quantum number means the orbital is used as 'Head of MTO'; lowercase means just used only as the 'tail of MTO'.

The switches for columns labeled as `occ` and `unocc`. take 0 (not included) or 1 (included). With the switch, we can construct two groups of orbitals, `occ` and `unocc`. In this sample `GWIN.V2` as for atom 1, $\{\phi_{4s}, \dot{\phi}_{4s}, \phi_{4p}, \phi_{4d}, \phi_{3d}^{\text{local}}, \phi_{3s}^{\text{core}}, \phi_{3p}^{\text{core}}\}$ consist the group `occ`, and $\{\phi_{4s}, \phi_{4p}, \phi_{4d}, \phi_{3d}^{\text{local}}, \phi_{4f}\}$ consists the group `unocc`. So the any product of combinations $\{\phi_{4s}, \dot{\phi}_{4s}, \phi_{4p}, \phi_{4d}, \phi_{3d}^{\text{local}}, \phi_{3s}^{\text{core}}, \phi_{3p}^{\text{core}}\} \times \{\phi_{4s}, \phi_{4p}, \phi_{4d}, \phi_{3d}^{\text{local}}, \phi_{4f}\}$ are included as for the basis of the product basis. As for atom 2, $\{\phi_{2s}, \phi_{2p}, \phi_{3d}\} \times \{\phi_{2s}, \phi_{2p}, \phi_{3d}, \phi_{4f}\}$ are included.

- Core section: (not worth to read, since we currently use no `CORE2`, `A=B=C=0`.)

Each line of the last section of `Product BASIS` forms

```
atom  1  n  occ unocc  ForX0 ForSxc :CoreState(1=yes, 0=no)
   1   2   1   A    x      B    C
```

At first you have to understand the concept of `CORE1` and `CORE2` in EQ.35 Ref.I. However, in our recent calculations, we do not use "CORE2" generally. So, in such a case, set `A=B=C=0`. And treat shallow cores (above `Efermi-2Ry` or so) as valence electron by "local orbital method" in `lmf`.

- Be careful. Current version is inconvenient... Need to repeat `mkGWIN.lmf2` to generate `GWinput` template when you add PZ (local orbital).

[(Note: you can skip here if you don't use CORE2.)

Each of **A**,**x**,**B**,**C** takes 0 or 1. There are some possible combination of these switches;

1. If you take (**A x B C**)= (1 0 1 1), then the core is included in core2. In other words, this core is treated in the same manner of the valence electron.
2. If you take (**A x B C**)= (0 0 0 0), then the core is included in core1. The (exchange only) self-energy related to this core is included in **SEXcore**. **C** is the key switch which determine whether it is included in core1 or core2. There could be another option.
3. If you take (**A x B C**)= (1 0 0 1). This core is in core2. But it is not included in the calculation of *D* and *W*. This core is only included for SEX and SEC calculations.

These three kinds of choices are reasonable ones but we can consider some another choice. In the following, we show how these switches (**A**,**B**,**C**) affect executions called from **gw_lmfh** (essentially as same as **gw_lmf**).

- **hbasfp0**(mode 3) :Product basis for exchange due to core.
We include the **C**=0 cores as a part of the product basis as if **A**=1 **x**=0.
- **hsfp0**(mode 3): exchange mode for core.
 Σ_x only due to the **C**=0 cores are calculated.
- **hbasfp0** (mode1): Product basis.
Only see the switch **A** and **x**. The product basis is generated from (occupied \times unoccupied), where **A**=1 core is included as one of the occupied basis.
- **hsfp0** (mode 1): exchange mode.
Only see the switch **C**. Σ_x due to valence and due to **C**=1 cores are calculated.
- **hx0fp0** (mode 1): $W - v$ calculation.
Only see the switch **B**. *W* is calculates using all the valence and **B**=1 cores.
- **hsfp0** (mode 2): correlation mode.
Only see the switch **C**. Σ_c due to valence and due to **C**=1 are calculated.
- After you perform **gw_lmfh** or anything, you find output files **lbas** by **hbasfp0** (mode1), and/or **lbasc** by **hbasfp0** (mode3) for core. These contains important information about how many and how product basis are chosen. E.g. '**grep nbloch lbas**' shows how many product basis are used in the calculations.

9.7 ANFcond (we can skip here since we do not check this option now. Need fix this if necessary.

This file is used in **hx0fp0** in the calculation of $W - v$ (or rather Π in the program) to specify the antiferro condition.

Note : Now only for the case that (a translation vector + spin flip) is a symmetry operation.

This should be given by hand. For the cases of not antiferro, this file should not exist. Even if **ANFcond** does not exists for antiferro case, **hx0fp0** works but it requires about two time computational efforts.

The existence of this file means the Antiferro condition is used for x0k
Product basis $B(\{\mathbf{r}\}-\{\mathbf{a}\})$ is translated to $B(\{\mathbf{r}\}-\{\mathbf{a}\}-\mathbf{Af})=B(\{\mathbf{r}\}-\{\mathbf{a}\}-\mathbf{T}_0)$
1d0 1d0 1d0 ! Af=Antiferro translation vector in Cartesian.
1 2
2 1
3 4
4 3

The first line specifies the Antiferro translation vector. From the second line, we specify that atom i in the primitive cell is mapped to what atom $j(i)$ in the cell with the opposite spin by the translation. In this case, $j(1) = 2, j(2) = 1, j(3) = 4, j(4) = 3$. You have to be careful as for the true atomic position used in the GW calculations can be different from the given atomic positions in `ctrl.MnO`. The true atomic positions is written in `LMTO`.

In the case of one-shot GW (`gw_lmf` and `gw_lmfh`), it may be better to set "up only" QPE, so that you only calculate QPE of up spins at the same time.

In the case of `gwsc`, we just calculate QPE for up spins automatically (QPNT section is neglected).

10 Main Output Files of GW part

10.1 QPU

This is the main output¹³ in human readable format.

An example of one-shot GW by **gw.lmfh** is (In the case of QSGW, Z ($Z = 1$) is not shown):

```
=====
quasiparticle energies MAJORITY
=====
E_shift=  0.4263273221017709D+00  0.6075150850568627D+00  0.7046628446164018D+00 eV

      q      state SEx  SEscore SEc   vxc   dSE  dSEnoZ  eLDA   eQP  eQPnoZ  eHF  Z   2Z*Simg  ReS(elda)
0.0 0.0 0.0 1  -29.56  -1.97  10.40 -20.22 -0.52  -0.90 -19.08 -19.42 -19.71 -30.81 0.58 0.95  -21.12
0.0 0.0 0.0 2  -30.52  -2.24  10.09 -21.53 -0.70  -1.14 -18.06 -18.58 -18.93 -29.72 0.61 0.96  -22.66
0.0 0.0 0.0 3  -20.67  -1.87   5.97 -16.85  0.19   0.28  -7.20  -6.83  -6.65 -13.32 0.67 0.66  -16.57
...
```

From the 6h line, we have the eigenvalue data. All of the unit of energy is in eV. We should note that the zero-level of these values **eLDA eQP eQPnoZ** can be changed by **hqpe**. This **eLDA - E_shift** are the eigenvalues relative to a Fermi energy determined by the smearing method. Detailed value of **eLDA** is in **TOTE2.UP**. Detailed value of **eLDA- E_shift** is in **TOTE.UP**.

q : **k** vector
state: Band index n , which is from the lowest eigenvalue (not include cores).
SEx: $= \langle \Psi_{\mathbf{k}n} | \Sigma_{\mathbf{x}}^{\text{core2+valence}}(\mathbf{r}, \mathbf{r}') | \Psi_{\mathbf{k}n} \rangle$
SEscore: $= \langle \Psi_{\mathbf{k}n} | \Sigma_{\mathbf{x}}^{\text{core1}}(\mathbf{r}, \mathbf{r}') | \Psi_{\mathbf{k}n} \rangle$
SEc: $= \langle \Psi_{\mathbf{k}n} | \Sigma_{\mathbf{c}}^{\text{core2+valence}}(\mathbf{r}, \mathbf{r}', \epsilon_n(\mathbf{k})) | \Psi_{\mathbf{k}n} \rangle$
vxc: LDA exchange correlation energy. $\langle \Psi_{\mathbf{k}n} | V_{\text{xc}}^{\text{LDA}}([n_{\text{total}}], \mathbf{r}) | \Psi_{\mathbf{k}n} \rangle$
dSE: $Z_{n\mathbf{k}} \times \text{dSEnoZ}$
dSEnoZ: $\langle \Psi_{\mathbf{k}n} | \Sigma_{\mathbf{x}}^{\text{core1}}(\mathbf{r}, \mathbf{r}') + \Sigma_{\text{xc}}^{\text{core2+valence}}(\mathbf{r}, \mathbf{r}', \epsilon_n(\mathbf{k})) | \Psi_{\mathbf{k}n} \rangle - \langle \Psi_{\mathbf{k}n} | V_{\text{xc}}^{\text{LDA}}([n_{\text{total}}], \mathbf{r}) | \Psi_{\mathbf{k}n} \rangle$
 $= \text{SEx} + \text{SEscore} + \text{SEc} - \text{vxc}$
eLDA: LDA eigenvalues. $\epsilon_n(\mathbf{k})$
eQP: QP energy. $\epsilon_n(\mathbf{k}) + \text{dSE}$
eQPnoZ: QP energy without Z . $\epsilon_n(\mathbf{k}) + \text{dSEnoZ}$
eHF: HF energy of 1st iteration. $\epsilon_n(\mathbf{k}) + \text{SEx} + \text{SEscore} - \text{vxc}$
Z: Z factor. $Z_{n\mathbf{k}}$
2Z*Simg: Quasi-particle life time. $2Z_{n\mathbf{k}} \times \text{Im} \langle \Psi_{\mathbf{k}n} | \Sigma_{\mathbf{c}}^{\text{core2+valence}}(\mathbf{r}, \mathbf{r}', \epsilon_n(\mathbf{k})) | \Psi_{\mathbf{k}n} \rangle$
(Is this really the usual definition of the life time?—don't believe me)
ReS(elda): $\text{Re} \langle \Psi_{\mathbf{k}n} | \Sigma_{\mathbf{x}}^{\text{core1}}(\mathbf{r}, \mathbf{r}') + \Sigma_{\text{xc}}^{\text{core2+valence}}(\mathbf{r}, \mathbf{r}', \epsilon_n(\mathbf{k})) | \Psi_{\mathbf{k}n} \rangle$

10.2 XCU

LDA exchange-correlation. Detailed data of above **vxc**.

10.3 SEXU

Exchange part of the self-energy due to valence electrons. Detailed data of above **SEx**.

10.4 SEXcoreU

Exchange part of the self-energy due to core. Detailed data of above **SEscore**.

¹³Note that QPU also implies QPD and so on. U is for up D is for down spins.

10.5 SECU

Correlation part of the self-energy. Detailed data of above SEc.

10.6 TOTE.UP (TOTE.DN)

This is a central output. It contains LDA and QP energies. These values are relative to a Fermi energy determined by the smearing method. It contains two kind of QP energies QP QPnoZ. The first line contains the Fermi energy in Ry determined by the smearing method. It is also shown in the end of DOSACC.la.

10.7 TOTE2.UP (TOTE2.DN)

This is a central output. It contains zero-level shifts from TOTE.UP. The first line contains the Fermi energy in eV (= the Fermi energy in TOTE.UP but it is in Ry) and three energy shifts E_shift, which are the same values in the 4th line of QPU.

Note that all *.chk files are just to check calculations (not read in by successive executions).

10.8 DOSACC.la

This lists all the eigenvalues in ascendant order. States with almost the same eigenvalues are degenerated states. The 4th column contains number of electrons up to the eigenvalue.

10.9 DOSACC2.la

This is similar with DOSACC.la. But we remove the degeneracy.

10.10 Core.ibas*_l*.chk

Used core eigenfunctions.

10.11 VXCFP.chk

This contains eigenvalues and $\langle \psi_{\mathbf{k}n} | V_{xc} | \psi_{\mathbf{k}n} \rangle$ in both units, Ry and eV. See below.

10.12 The Fermi energies in this GWcode.

We mainly have two kinds of Fermi energy $E_{\text{FEERMI}}^{\text{smear}}$ $E_{\text{FEERMI}}^{\text{tetra}}$.

1. At first eigenvalues given by **lmfgw** is in VXCFP.chk. You can see

```
%head VXCFP.chk
### LDA exchange correlation ###
#   qvec           ikp iband   eigen      VXC(ntotal)    VXC(nvalence)   eigen(eV)   VXC
  0.0000  0.0000  0.0000  1  1    -0.96932423    -1.00727912    0.00000000   -13.18843159  -
...
```

These are raw values. TOTE contains the eigenvalues but relative to a Fermi energy $E_{\text{FEERMI}}^{\text{smear}}$ which is determined by the smearing method. It is also shown at the top part of output files lsx.sf and lsc.sf. And you also see the value at the end of DOSACC.la.

This is the head of TOTE.UP;

%head TOTE.UP

43	8	8.520283353474250E-003						
0.0000000	0.0000000	0.0000000	1	1	-0.1330435686590073D+02	-0.1322984339282777D+02	-0.1	
0.0000000	0.0000000	0.0000000	2	1	-0.7555264915356062D+00	-0.6267595395613325D+00	-0.6	
...								

Here $E_{\text{FEERMI}}^{\text{smear}}=8.520283353474250\text{E-}003$. From the second lines, they are LDA eigenvalues and QP energies (Z included and Z=1); they are relative to the $E_{\text{FEERMI}}^{\text{smear}}$.

-13.18843159 eV - $E_{\text{FEERMI}}^{\text{smear}}$ (which should be translated into in eV) = -0.1330435686590073D+02 eV. Here -13.18843159 is the value in VXCfp.chk shown above.

2. There is the another Fermi energy $E_{\text{FEERMI}}^{\text{tetra}}$, which is used by mode 11 (or mode 1) of `hx0fp0` in `gw_lmfn`. It is determined by `heftet` and stored in `EFERMI`.
3. `hqpe` gives `TOTE2.UP` and `QPU`. They contains the same values. You can see `eLDA eQP eQPnoZ Z` not only in `QPU` but also in `TOTE2.UP`. At top lines of `TOTE2.UP`, you see

%head TOTE2.UP

43	8	0.1159252712507000D+00	0.7555207081466229D+00	0.6267572296579150D+00	0.60094
0.0000000	0.0000000	0.0000000	1	1	-0.1254883615775411D+02
0.0000000	0.0000000	0.0000000	2	1	-0.5783388983382487D-05
0.0000000	0.0000000	0.0000000	3	1	-0.1369098933889923D-05
0.0000000	0.0000000	0.0000000	4	1	0.0000000000000000D+00
...					

,where a number in first line $E_{\text{FEERMI}}^{\text{smear}}=0.1159252712507000\text{D}+00$ eV = 8.520283353474250E-003 Ry, the same as the previous one. This is a case when you did `hqpe` with `augment 4` (it means we set the 4th-band eigenvalue zero). Another 3 values in the first line are shifts from `TOTE`. Shown `eshift(eLDA) = 0.7555207081466229D+00` eV. E.g., the second line shows
-0.1254883615775411D+02 eV = -0.1330435686590073D+02(in `TOTE`) + `eshift(eLDA)` eV.

When you do `hqpemetal`, three shifts at the first line in `TOTE2.UP` is determined so as to give the eigenvalues relative to the Fermi energies shown in `EFERMI`, `EFERMI.QP1`, and `EFERMI.QPz=1`. These are Fermi energies by tetrahedron method.

As for `gwbands_lmfn`, it recalculates eigenvalues for all **q** along SYML. Then the default "zerolevel" = $E_{\text{FEERMI}}^{\text{smear}} - \text{eshift}(\text{lda})$. Because the eigenvalues given by this band-mode are presumably the same, we have

Shown LDA eigenvalue

= -13.18843159(raw data by band mode—same as that in VXCfp.chk) - zerolevel
= (-13.18843159 - `EFERMIsmear`) + `eshift(lda)`.
= -0.1330435686590073D+02(this is in `TOTE.UP`) + `eshift(lda)`
= -0.1254883615775411D+02(this is in `TOTE2.UP`).

It means that values in `TOTE2.UP` recovers. But if raw data by band mode is different from it, these is a trouble. It does not recover the values in `TOTE2.UP(=QPU)`.

As for the QPE, we calculate the difference from LDA values in `TOTE2.UP` at first, and add the difference to the Shown LDA eigenvalue.

11 mkGWIN_lmf2 and its I/O Files

(QPNT.chk contains irreducible k point for given n1 n2 n3; KPTin1BZ.gwinit.chk contain all k points in Brillouin Zone).

The purpose of the script **mkGWIN_lmf2** is to give a template **GWinput.tmp**. The script is complicated because of historical reasons. However, its essential is simple; we calls three executions in this script as

echo 0 | lmfgw si

echo 1 | gwinit

echo -100 | qg4gw

. We explain each by each.

11.1 echo 0|lmfgw

— makes SYMOPS LATTC CLASS NLAindx.

Input files

- **GWINO** : This is a file, which contains your supplied n1 n2 n3 when you invoke the script. This file is given within the script of mkGWIN_lmf2 (as "here document").

```
cat <<EOF >GWINO
n1 n2 n3
  $n1 $n2 $n3
cut
  4.0 3.0
alpha
  1
Number of bands
  999 99999.0
  999 3.0
EOF
```

- **ctrl.si** : Master input file of lmf calculation.

Output files

- **LATTC** : contains the information of primitive translation vectors, lmxa and konf. See ??
- **SYMOPS** : The point group operations. See ??
- **CLASS** : Equivalent atomic positions are called as 'class'. This small file contains a map between atomic site and 'class'.
- **NLAindx** : This file contains indexes (p_{valence}, l, a) for orbitals in the MT. (p_{valence} is radial function index, a is atomic site index). Eigenfunctions are expanded in this order.
- **ldima** : Number of MTOs for each atomic site. (this is used only from **hqpe_sc**—QSGW mode).
- **ves*** : not meaningful at this stage
- **rhoMT*** : not meaningful at this stage

11.2 gwinit

— Get GWIN_V2.tmp and QPNT.tmp

Input files

- **GWINO** :
- **LATTC** :
- **SYMOPS** :
- **NLAindx** :

Output files

- **GWIN_V2.tmp** : A part of GWinput.tmp
- **QPNT.tmp** : A part of GWinput.tmp
- (**KPNTin1BZ.gwinit.chk**) : check KPNT in the 1BZ.

If SYML exist, **gwinit** gives also a template **QPNTforSYML.tmp** suitable for such SYML. Here SYML specify how to plot the energy band. See explanation for **bandplot** script.

Note that LATTC SYMOPS CLASS NLaindx are overwritten when you execute **gw_lmfh** because we repeat **echo 0|lmf** at the head of **gw_lmfh**.

11.3 echo -100|qg4gw

— Generate GWinput.tmp

Input files

- GWIN0 : (copy of GWIN0.tmp by **gwinit**)
- GWIN_V2 : (copy of GWIN_V2.tmp by **gwinit**)
- QPNT : (copy of QPNT.tmp by **gwinit**)

Output files

- GWinput : (this is copied to GWinput.tmp)

This command "echo -100|qg4gw" is a file converter from these two files into GWinput. And it is copied to GWinput.tmp. (**mkGIN_lmf** keeps GWinput if it exist before you invoke it.).

12 gwsc script and its I/O Files

In **gwsc**, we have a loop of QSGW self-consistency. Look into the **gwsc** script. In each iteration, we perform these fortran programs;

```
NO_MPI=0 #this is used for non-mpi versions of fortran program.

### self-consistent calculation for given Sigma(self-energy) ###

    run_arg '---' $MPI_SIZE $nfpwgw /lmf-MPIK    llmf $TARGET

### Preparation stage #####

argin=0; run_arg $argin $NO_MPI $nfpwgw /lmfgw      llmfgw00 $TARGET
argin=1; run_arg $argin $NO_MPI $nfpwgw /qg4gw      lqg4gw  #Generate requied q+G v
argin=1; run_arg $argin $MPI_SIZE $nfpwgw /lmfgw-MPIK llmfgw01 $TARGET
        run_arg '---' $NO_MPI $nfpwgw /lmf2gw      llmf2gw  #reform data for gw

### Main stage of gw #####

argin=0; run_arg $argin $NO_MPI $nfpwgw /rdata4gw_v2 lrdta4gw_v2 #prepare files
argin=1; run_arg $argin $NO_MPI $nfpwgw /hefttet     lefttet # A file EFERMI for hx0fp0
argin=1; run_arg $argin $NO_MPI $nfpwgw /hchknw      lchknw # A file NW, containing nw

## Core part of the self-energy (exchange only) ##

argin=3; run_arg $argin $NO_MPI $nfpwgw /hbasfp0     lbasC # Product basis generation
argin=3; run_arg $argin $MPI_SIZE $nfpwgw /hvccfp0    lvccC # Coulomb matrix for lbasC
argin=3; run_arg $argin $MPI_SIZE $nfpwgw /hsfp0_sc   lsxC # Sigma from core1

## Valence part of the self-energy Sigma ##

argin=0; run_arg $argin $NO_MPI $nfpwgw /hbasfp0     lbas # Product basis generation
argin=0; run_arg $argin $MPI_SIZE $nfpwgw /hvccfp0    lvcc # Coulomb matrix for lbas
argin=1; run_arg $argin $MPI_SIZE $nfpwgw /hsfp0_sc   lsx # Exchange Sigma
argin=11; run_arg $argin $MPI_SIZE $nfpwgw /hx0fp0_sc lx0 $lx0_para_option #x0 part
argin=2; run_arg $argin $MPI_SIZE $nfpwgw /hsfp0_sc   lsc #correlation Sigma
argin=0; run_arg $argin $NO_MPI $nfpwgw /hqpe_sc      lqpe #all Sigma are combined.

run_arg:
```

Here a subroutine of bash **run_arg** was used, which is given in **ecalj/lm7K**; it just invoke a command with the argument **\$argin** (this is read by **read(*,*)** in fortran). In cases with **MPI_SIZE/=0**, **mpirun** is invoked. Console out put go to **l*** files. For example,

```
    argin=2; run_arg $argin $MPI_SIZE $nfpwgw /hsfp0_sc lsc #correlation Sigma
```

invokes **hsfp0_sc** with argument '2' by **mpirun** with the **-np \$MPI_SIZE**. Console outputs are written into logfiles such as **lqpe**. **\$nfpwgw** contains path to the execution binaries.

In the following, We explain input/output files for each fortran program. Note that “**echo 0|lmfgw**” means invoking **lmfgw** with **argin=0**.

12.1 echo 0| lmfgw si

See Sec.11.1.

12.2 echo 1| qg4gw

This makes **q** points, and **G** vectors for these **q**. (**q** was **k** in previous sections.) Main routine of qg4gw is `fpgw/main/qg4gw.m.F` and calls `fpgw/gwsrsc/mkqg.F`

Input files

- GWinput :
- LATTC :
- SYMOPS :

Output files

- QGpsi : (bin) q and G vector for the eigenfunctions.
- QGcou : (bin) q and G vector for the Coulomb matrix
- Q0P : offset- Γ points which are the replacement of the q=0 points. See section??.
- QIBZ : q points in the Irreducible BZ.
- BZDATA : (bin) BZ data for integration (include tetrahedrons if necessary). See e.g. `main/hx0fp0.sc.F` and search "call read_BZDATA", which is a readin routine of this file defined in `rwzbzdata.F`.
- KPTin1BZ.mkqg.chk : list of q in the 1st BZ for check.
- QBZ : q point in the 1st BZ.
- EPSwklm : Required information for the BZ integration (mainly in order to evaluate the weight in the Γ cell). See Eq.xxx in [?].

12.3 echo 1|lmfgw si

Calculate eigenfunctions, eigenvalues and $\langle\psi|H_{KS}|\psi\rangle$

Input files

- ctrl.si :
- rst.si : (bin) Restart file of the lmf calculation. It contains all information
- sigm.si : (bin) If this exist and
- QGpsi, QGcou, Q0P : :
- NLAindx : :

Output files

- gwa.si : (bin) atomic data
 - gwb.si : (bin) band data
 - gw1.si : (bin) $\langle\psi|H_{KS}|\psi\rangle$
 - gw2.si : (bin) $\langle\psi|H_{KS} - V_{xc}(n_{total})|\psi\rangle$.
 - vxc.si, evvec.si : (bin) used in `hqpe.sc.m.f` as "v_xc" and "evvec").
- vxc.si contains $\langle\psi|V_{xc}(n_{total})|\psi\rangle$ including off-diagonal part. evvec.si contains eigenfunctions.
- normchk.si : norm check (only for check) This is like this

```
> head -20 normchk.si
#      IPW      IPW(diag)      Onsite(tot)      Onsite(phi)      Total
      0.436015      0.805123      0.563972      0.562573      0.999988
      0.339134      0.620353      0.660515      0.656881      0.999649
      0.339133      0.620353      0.660516      0.656882      0.999649
      0.339133      0.620353      0.660516      0.656882      0.999649
      0.507738      0.648515      0.492040      0.487673      0.999778
...
```

This check is sometimes important for debugging and to determine the cutoff parameter

QGcut_psi

. The first line (corresponding to 1st band of 1st q point) means that total normalization almost unity = 0.999988 = 0.436015 + 0.563972. Because we expand the MTO by IPW, the normalization is a bit different from unity, especially for higher bands. You can see that it get closer to unity for larger QGcut_psi, though it does not reach to unity because of some contribution of the higher angular momentum contribution within MT. [Values of Onsite(phi) are not correctly shown in the case when you use local orbital.]

Due to historical reason, data in vxc.si and exec.si and others contains duplicated data.

12.4 lmf2gw

All the required information are stored into DATA4GW_V2 and CphiGeig.

Input files

- gwa.si :
- gwb.si :
- gw1.si :
- gw2.si :
- Q0P :
- CLASS :
- NLAindx :

Output files

- DATA4GW_V2 : (bin) Main data for GW calculations.
I/O of DATA4GW_V2 is controlled by `gwinput.f`, which contains detailed information.
- CphiGeig : (bin) Eigenfunctions for GW calculations.
- VXCfp.chk : Eigenvalue and Vxc check (only used for check)
It is like this;

```
### LDA exchange correlation ###
#   qvec           ikp iband   eigen      VXC(ntotal)    VXC(nvalence)    eigen(eV)    VXC(ntotal)(eV)    VXC(nvalence)
0.0000 0.0000 0.0000 1 1    -0.68505346    -0.91850436    0.00000000    -9.32070032    -12.49698668    0.00000000
0.0000 0.0000 0.0000 1 2     0.19292662    -0.99853478    0.00000000    2.62492096    -13.58586453    0.00000000
0.0000 0.0000 0.0000 1 3     0.19292763    -0.99853469    0.00000000    2.62493477    -13.58586334    0.00000000
0.0000 0.0000 0.0000 1 4     0.19292777    -0.99853461    0.00000000    2.62493664    -13.58586222    0.00000000
...
```

Here `VXC(nvalence)` is not used now. The eigenvalue in `eigen` is in Ry.

— This is the end of the preparation stage. —

From here, the main stage.

12.5 rdata4gw_v2

— Read DATA4GW_V2 and some files, and decompose it into files required in the following GWsteps. (checked! dec2014)

Input files

- GWinput :
- DATA4GW_V2 :
- CphiGeig :
- QGpsi :
- QGcou :
- Q0P :
- QIBZ :
- SYMOPS : points group operations.

Output files

- hbe.d : data size
- Core.ibas*_l*.chk : core eigenfunctions just for check.
- LMTO : basic data for the crystal.
- EValue : (bin) valence eigen value
- ECORE : core data and core eigenvalues
- CPHI : (bin) Coefficients of eigenfunctions as for the atomic-like argumentation waves in MTs'.
- GEIG : (bin) Coefficients of eigenfunctions as for IPW.
- PHIVC : (bin) All the radial functions.
- @MNLA_CPHI : index set for CPHI. This is not refereed just a check write.
- @MNLA_core : index set for core. This is not refereed just a check write.
- VXCfp : (bin) this is for diagonal elements of $V_{xc}^{LDA}(n_{total})$.
- PPOVLI.* : (bin) Overlap matrix of IPW. xxxxxxxxxxxx

- PPOVLG.* : (bin) PPOVLG Overlap matrix xxxxxxxxxxxxf IPW. not exactly the the overlap matrix. see around line 500 in rdata4gw.m.f
- PPOVL0 : (bin) xxxxxxxxxxxxxx
- HVCCIN : (bin) Required inputs for hvccfp0. Information in this files.
- NQIBZ : q point info. Only used for parallel test mode.
- normchk.dia : Norm check. These numbers should be almost the same as those in normchk.si

These files are input for the following steps. The name of file *fooU* means that it relates to up-spin. We have *fooD* files in the case of spin-polarized calculation with **nspin=2**.

12.6 echo 1|heftet

— Get the Fermi energy EFERMI by tetrahedron method. It is used in **hx0fp0**.

Input files

- EVU :
- BZDATA :
- GWinput :
- ECORE : (dummy)
- SYMOPS : (dummy)
- LMTO :
- hbe.d :

Output files

- EFERMI : contains Fermi energy given by the tetrahedron method. It is used in **hx0fp0** but not in **hsfp0**.
- DOSACC.lda,DOSACC2.lda : They are lists of the all the eigenvalues from the bottom. DOSACC2.lda is a list to show only the un-degenerated eigenvalues. They are just check write. But it is an indicator for you to determine **esmr** in GWinput.

12.7 hchknw

— Calculate the required number of ω points along real axis.

This NW is not essentially used in **gw_lmfh** (but required as a dummy file). Only used in **gw_lmf**.

Input files

- BZDATA :
- GWinput :
- ECORE : (dummy)
- SYMOPS : (dummy)

Output files

- NW : contains number of ω points.

12.8 echo 3|hbasfp0

— Make product basis.

Mode 3 is for the core states. It generate a product basis on each MT suitable to expand to calculate the exchange part due to core. See explanations for the input file of GWinput.

Input files

- LMTO :
- PHIVC :
- GWinput :

Output files

- BASFP* : (bin) Product basis functions
- PPBRD_V2_* : (bin) Radial integrals on each MT, symbolically written as $\int \phi(r)\phi(r)B(r)dr$
- PHIV.chk : Valence radial functions (for check).

12.9 echo 0|hvccfp0

— Calculate the Coulomb matrix in the Mixed basis

Input files

- HVCCIN :
- Q0P :
- BASFP* :

Output files

- VCCFP : The Coulomb matrix expanded in the mixed basis
- Mix0vec : This is used only for dielectric-constant calculation (mode 2 or 3 of **hx0fp0**). This contains the expansion of the plane wave $\exp(i\mathbf{q}\mathbf{r})$ in the mixed basis. See Usuda's note.

12.10 echo 3|hsfp0

— Exchange part of the self-energy for the core

Input files

- GWIN_V2,LMTO,ECORE :
- CLASS :
- hbe.d :
- Q0P :
- PPBRD_V2_* :
- CPHI :
- GEIG :
- VCCFP :
- PPOVL :

Output files

- SEXcoreU : The core part of the exchange self-energy for \mathbf{q} and band index specified in <QPNT>. See 10.

12.11 echo 0|hsfp0

— Make product basis for the valence part.

12.12 echo 1|hsfp0

— Exchange part of the self-energy for the valence part.

Output files

- XCU : The LDA exchange self-energy for \mathbf{q} and band index specified in <QPNT>. See 10.
- SEXU : The valence part of the exchange self-energy for \mathbf{q} and band index specified in <QPNT>. See 10.

12.13 echo 11|hx0fp0

— Screened Coulomb interaction W (Sergey mode)

Input files

- GWinput, LMTO, Ecore, EVU :
- NW : dummy
- hbe.d :
- Q0P :
- PPBRD_V2_* :
- CPHI,GEIG :
- PPOVL :
- VCCFP :
- ANFcond : (optional) This file is to specify antiferro condition. This should not exist for other cases. This file should be given by hand.

Output files

- WV.d : size of the dielectric function
- WVR : (bin) $W - v$ in the expansion of mixed basis along the real axis

- WVI : (bin) $W - v$ in the expansion of mixed basis along the imaginary axis

12.14 echo 12|hsfp0

— Correlation part of the self-energy(Sergey mode)

Input files

- GWinput, LMTO, Ecore, SYMOPS : These are readin by **genallcf_v3**.
- CLASS, hbe.d, EVU, Q0P :
- PPBRD_V2_* :

Radial integrals on each MT, symbolically written as $\int \phi(r)\phi(r)B(r)dr$. These are generated by **hbasfp0**.

- CPHI,GEIG :
- PPOVL :
- WV.d, WVR, WVI :

Output files

- SECU : The correlation part of the self-energy for **q** and band index specified in <QPNT>. See 10.

12.15 echo 0|hqpe

— Summarize the output

Input files

- SEXcoreU,XCU,SEXU,SECU : See 10.

Output files

- QPU : The QP energies and related value summary in human interface. See 10.
- TOTE : The detailed values of the QP energies. See 10.
- TOTE2 : The detailed values of the QP energy. See 10. This is used for **bndplot**.

NOTE: For example, if you do **{echo 4\$|hqpe}{hqpe}**, it just shift zero level of QPE, so that 4th line (counted from top) eigenvalue (in QPU) is to be zero.

13 Check list for convergence on GW calculations

Results could be dependent on cutoff parameters in **GWinput**. In my opinion, generally speaking, it is so easy to have convergence more than $\sim 0.1\text{eV}$ for band gap...

- Number of k points `nln2n3`.

Probably $4 \times 4 \times 4$ (or $6 \times 6 \times 6$) are reasonable choice for insulator in the case of two atoms such as GaAs. In other words, “used periodic cell volume” = $4 \times 4 \times 4 \times (\# \text{ of atoms}) \times (\text{Volume per atom})$. For example, for $2 \times 2^3 = 16$ atoms per cell, we can use $2 \times 2 \times 2$ instead of $4 \times 4 \times 4$ (this is the same as in the case of LDA).

For metals, $12 \times 12 \times 12$ or more per atom may be required. But it depends on case by case.

- MTO’s and APWs. Basis for eigenfunctions.

If we like to get “best converged results, we may need to use large enough MTO’s. (I think the default setting is reasonable; but there is a room to change MTO settings in `ctrl.*`). But, in cases, we can not use large APW cutoff (`pwemax`) more than $3.0 \sim 4.0$ Ry because of poor linear dependency of basis set (calculation in LDA level fails). In such a case, we need to use “smaller MT radius `R=`”. Then we may need semi-core as local orbital when “its spillout in the outside of MTs is too large”.

In **GWinput**, we can set the number of unoccupied states which you take into account by `emax_chi0`, `emax_sig`, `nband_chi0`, and `nband_sig`. But we usually unset them except `emax_sig` for **gwsc**.

(We may need a kind of completeness of the basis set; the completeness could be important from the view of ‘Coulomb hole’ picture.)

NOTE: Current PMT-QSGW method [?] expand the static version of self-energy of QSGW just in the basis of MTO’s (no APWs). Thus the expansion can be unsatisfactory (it depends on case by case, and required convergence). In such a case, we inevitably have to use empty spheres (MTO’s) which is for empty region.

- Cores.

We usually use all cores as **core1** (exchange only core). If necessary, it is better to treat shallow cores by the local orbitals (such cores are treated as valence).

If we treat cores by **core2** (not only for exchange, but also in dielectric functions), we have to be careful about the core wave orthogonalization with respect to valence eigenfunctions; this is a little complicated; probably it is better not to use **core2**).

(This is related to `CoreOrth` (only for core2). If it affects so much, *D* function might be too poor due to the poor orthogonality condition between core and valence.) (NOTE: `CoreOrth` is not maintained recently)

- `QpGcut_psi` IPW cutoff to expand eigenfunctions in the interstitial region. We usually use `QpGcut_psi 4.0`. Usually not so bad. Larger is better but expensive. You may test calculations with `QpGcut_psi 3.0`, and how much difference of results.
- `QpGcut_cou` IPW cutoff to expand the Coulomb interaction in the interstitial region (Mixed product basis(MPB) consist of this IPW and PB). interstitial region. We usually use `QpGcut_cou 3.0`. Usually not so bad. Larger is better but expensive. You may test calculations with `QpGcut_cou 2.5`, and how much difference of results.

- Product basis section.

At least, `lcutmx=4` will be necessary for atoms with d electrons. (but `lcutmx=2` look reasonable for oxygen, for example).

- `esmr`.

In our experience, `esmr=0.003000` (default) is reasonable. But there is a room to check stability on it for metals. In principle, for larger `nln2n3`, we can use smaller `esmr`.

- `dw, omg_c niw`

It will be worth to try to check how much the results changed due to them. But usually `dw=0.005, omg_c=0.04` is not so bad a choice. As for `niw=10` seems to be not so bad usually, but it is safer to check the convergence on it (test cases with `niw=6,10,12,16`).

- `deltaw`. ~ 0.01 a.u. will be not so bad. See two Z values shown in SXCU. It is better to try to check how about the dependence on this.
- `chi_regqbz off`. (on is default). If off, we use off-Gamma mesh (Gamma point is between mesh points) for dielectric functions when we perform GW or QSGW. “chi_regqbz off” may accelerate convergence on number of k points.

14 Linear response calculations

With these scripts for linear response calculations, **eps***, we can calculate **q**-dependent dielectric function $\epsilon(\omega, \mathbf{q})$ (and v , W) (and χ for spin fluctuation). But (because of numerical reason), we can not use $\mathbf{q} = 0$ limit. (if $|\mathbf{q}|$ is too small, we have numerical problem, zero divided by zero, because we have not implemented the version to use $\mathbf{q} = 0$.)

- **eps_lmfh**
Dielectric function epsilon with local field correction. Expensive calculation (we may need to reduce number of wing parts in future...).
- **epsPP_lmfh**
epsilon without local field correction. $1 - \langle e^{i\mathbf{q}\mathbf{r}} | v | e^{i\mathbf{q}\mathbf{r}} \rangle \langle e^{i\mathbf{q}\mathbf{r}} | (\chi^0) | e^{i\mathbf{q}\mathbf{r}} \rangle$
- **epsPP_lmfh_chipm**
For spin susceptibility. This essentially calculate non-interacting spin susceptibility. Then it is used for the calculation of full spin susceptibility with **util/calj_*.F** programs (small quick programs). See spin wave paper. See spin susceptibility section Sec.??.
- (not maintained now; we will recover this) **eps_lmfh_chipm**
This gives full non-interacting spin susceptibility. Testing. We have to determine U (Stoner I) for the determination of full spin susceptibility. TDLDA? or so?
- (This is old mode --- not maintained) **epsPP_lmfh_chipm_q**
For spin susceptibility, spin susceptibility $\langle e^{i\mathbf{q}\mathbf{r}} | \chi(q, \omega) | e^{i\mathbf{q}\mathbf{r}} \rangle$ In this script, You have to assign that `isp=1` is majority, `isp=2` is minority. This is with long wave approximation.

- We use the histogram method (the Hilbert transformation method); we first calculate its imaginary parts with the tetrahedron technique for dielectric functions. Then we get its real part by the Hilbert transformation.

You need to choose `[dw, omg_c]`. The width of histogram bins are getting larger when omega gets larger. `dw` is the size of histogram-bin width at $\omega=0$. At $\omega=\text{omg_c}$, its width gets twiced.

To plot dielectric function with reasonable resolution, it might be better to set `dw 0.001` and `omg_c 0.1` for example. You may have to choose small enough omega for spin wave mode as 0.001 Ry (Or smaller). `omg_c` is given like 0.05 Ry or so. But sometimes it can be like 1Ry.

- **epsPP_lmfh** only calculation an matrix element of dielectric function for $\exp(i\mathbf{q}\mathbf{r})$. Thus very faster than **eps_lmfh** mode. It uses a special product basis set for cases without inversion (problem is in how to expand $\exp(i\mathbf{q}\mathbf{r})$ in the MPB; the product basis is not from ϕ and ϕ_{hidot} , but from spherical Bessel functions).

In `*_lmfh_*` modes(I now use little for `*_lmf_*` modes), you can use small enough delta. Use small enough delta ($\approx 10^{-8}$ a.u.) for spin wave modes (also you can use it for dielectric function and GW). This is necessary because pole is too smeared if you use larger delta.

14.1 eps_lmfh, epsPP_lmfh: the dielectric functions

You can invoke the script, e.g. as `"eps_lmfh si"`.

Specify **q** point in `<QforEPS>` or so. Mesh for ω is specified by `[dw, omg_c]`.

The obtained data are in `EPS*.dat` and `EPS*.nlfc.dat`. `EPS*.nlfc.dat` contains the result without local-field correction `EPS*.dat` contains the result with local-field correction (this is generated only for **eps_lmfh**. Both of them contains

`q(1:3)`, ω , $\text{Re}(\epsilon)$ $\text{Im}(\epsilon)$, $\text{Re}(1/\epsilon)$, $\text{Im}(1/\epsilon)$

in each line.

- This code works OK only for **q** is near 0. Be careful for $\mathbf{q} \rightarrow 0$ limit. Too small **q** can give strange spectrum at high energy (real part is affected by it)

Because $\mathbf{q} \rightarrow 0$ gives too large cancellation effects (the denominator and numerator go to zero—it means we need very accurate orthogonalization between occupied and unoccupied states). This is a kind of disadvantage of our method (though there is an advantage—our code can calculate dielectric function even for metal as far as you use large enough number of \mathbf{k} point.)

- The calculate of dielectric functions usually requires so many k point. For example, for Si, $n_1\ n_2\ n_3 = 4\ 4\ 4$ is too small. It gives too large dielectric constants ~ 19.4 though the converged value should be ~ 13 . (we need $10 \times 10 \times 10$ or more like $20 \times 20 \times 20$ for some reasonable results). For GaAs, we observed that reasonable $\epsilon(\omega)$ requires rather large number of \mathbf{q} points like $15 \times 15 \times 15$ or $20 \times 20 \times 20$ for `[n1n2n3]`. This is too time-consuming to get result (but you can use “very small product basis”(just sp polarization for this purpose; it makes speed up so much). Or, you can calculate “ $\epsilon(\omega)$ without LFC”. See section for `eps_PP_lmfh`.

- Core orthogonalization problem (only when core2 is used)
—`Core0rth` is not maintained recently — `Core0rth` gives so serious effect for $\epsilon(\omega)$, if you include some cores as “core2” in the product basis setting. (This means that you includes transitions from “core2 to valence” in the calculation of $\epsilon(\omega)$).

Then you have to use “Core0rth on”. Without it, you will have rather large imaginary part at rather high energy. Such transitions from core to higher valence bands is artificial due to the incomplete orthogonality between core and the higher bands. However, shallower d semi-core might be deformed too much by this option. Try to plot `Core_*.chk` files, which contains core radial functions. Anyway, it is better to treat shallow core as valence by “local orbital”.

14.2 epsPP_lmfh: the dielectric function(No LFC— faster)

You can calculate ϵ without LFC by `epsPP_lmfh`. It is very faster than `eps_lmfh`.

To calculate $\epsilon(\mathbf{q}, \omega)$ without LFC accurately, the best basis set for the expansion of the Coulomb matrix within MT is apparently not the product basis, but the Bessel functions corresponding to the plane waves $\exp(i\mathbf{qr})$. We use such a basis in this mode. However, our experience shows that the changes are little even with the usual product basis (we don’t describe this here).

14.3 How to calculate correct dielectric function?

(this subsection is essentially OK... but need to clean it up. dec2014)

There are problems to calculate correct epsilon.

At first, we talk about `epsPP_lmfh`, which is No LFC. Main problem are

1. Convergence for number of k point (specified by `n1n2n3`).

Roughly speaking, $20 \times 20 \times 20$ is required for not-so-bad results for Fe and Ni.

It is better to do $30 \times 30 \times 30$ to see convergence check.

However, in the case of ZB-MnAs (maybe because of simple structure around Ef), it requires less q points.

figs are for GaAs.

fig001: `n1n2n3` convergence for `Chi_RegQbz = on` case.

fig002: `n1n2n3` convergence for `Chi_RegQbz = off` case.

(`Chi_RegQbz` is explained in General section in this manual).

As you see, k points convergence looks a little better in `Chi_RegQbz=off`

(mesh not including gamma). However a little problem is that its threshold around 0.5eV is too high and slowly changing.

fig003: Alouanis'(from Arnaud) vs. 'Chi_RegQbz = on' vs. 'Chi_RegQbz = off'
 As you see, the threshold of the Red line (20x20x20 Chi_RegQbz=on) and Alouani's
 are almost the same, but the red line is too oscillating at the low energy part.
 On the other hand, 'Chi_RegQbz = off' in Green broken line is not so satisfactory
 at the low energy part.

fig.gas_eps_kconf.pdf shows the convergence behavior of epsilon for

2. $q \rightarrow 0$ convergence (this is related to whether Chi_RegQbz=on or off).

If you use very small q like $q=0.001$ is GaAs, it can cause a problem.

Use $q=0.01$ or larger (maybe $q=0.02$ or more is safer).

Very small q can give numerical error for high-energy region.

In fig004, we show the high energy tail part of $\text{Im } \epsilon(\omega)$ for GaAs case.
 At $q=0.01$ (this means $q = 2\pi/a_{\text{lat}} * (0 \ 0 \ 0.01)$), the imaginary part
 is a little too large. Less than 80eV, $q=0.02$ gives good results when compared with
 other high q results, though it still has noise above 80eV.

In fig005, I showed the same results compared with Alouani's (his is up to 40eV).

Both gives rather good agreements. As you see, $q=0.06$ or above might be necessary
 to get reasonable convergence for high energy part above 40eV.

We have to be careful for this poorness in high energy part--- it may effect
 low-energy $\text{Re}[\epsilon]$ through KK relation. However this can be very small
 enough.

In fig.gas_eps_qconv.jpg, we checked the convergence of $\epsilon(\omega=0, q)$ for $q \rightarrow 0$.

As you see, it gives convergence, however, $q=0.01$ is a little out of
 curve---this should be because of the poorness in the high energy part.

so $q=0.02$ or $q=0.03$ is safer, and you can get ϵ within 1 percent accuracy.

3. Including Core for dielectric constant is dangerous.

It can cause very poor results if you include core part in $G_{\text{win}}^{\text{input}}$.

You need to include core just as valence (with local orbital).

In fig008, we showed core effects. It starts from $\sim 16\text{eV}$
 (this is core to conduction transition).

fig007 showd the check about the q point dependence---even with large q ,
 it would not change.

These shows that the core excitation can have larger energy range.

This is in contrast to the valence case

(then the most of excitaion is limited to less than 10eV).

We have to be careful for such high-energy exciation... The LMTO basis might
 be not so good for high energy.

4. basis set.

Use $Q_{\text{Gcut_psi}} \sim 3.0$ a.u. or so (as same as GW calculation).

In the case of ϵ_{PP}^* mode,

$Q_{\text{Gcut_cou}}$ can be very small--- In our codes now,

$\text{ngc} \geq 1$ should be for all q vector shown in lqg4gw02 (output of echo 2|qg4gw).

[In principle, it should be only for the q vector for which we calculate epsilon.

But there is a technical poorness in our code---

(maybe) a problem here; the plane-wave part of the eigenfunction generated
 in lmf2gw is not correctly passed to lmf2gw when $\text{ngc}=0$].

```
-- eps_lmfh: including LFC -----
To include eps with LFC, do eps_lmfh.
But lcutmx=2 seems to be good enough to get 0.5 percent error (maybe better than this).
Test it 10x10x10 or so. (I need to repeat if necessary).
Further you can use smaller QpGcut_cou like 2.2 or so,
with rather smaller product basis (up to p timed d, not including f).
```

Note: epsPP_lmfh is designed to use good basis to calculate eps without LFC. This is usually in agreement with what you obtained by eps_lmfh; however it can give slight difference when you use small product basis.

```
---Summary -----
```

So in conclusion, I think a best way to do is

1. set $q=0.02$ [$q=2\pi/\text{alat}(0\ 0\ 0.02)$] or so for GaAs case.
If you want to check, do $q=0.03$ and $q=0.06$ also.

“Chi_RegQbz = off” is better for materials like GaAs with direct gap.
2. You can use small QpGcut_cou but all ngc should be one or more.
3. As for the Product basis setting in epsPP* scripts, only lcutmx and tolerance (this can be like 0.001 or so) are relevant.
E.g. set lcutmx=4 or so.
4. Do nk=20 18 16 and take interpolation to determine $\text{eps}(\omega=0, q=0)$.
5. To get eps with LFC, set QpGcut_cut as xxx, and set lcutmx=2 where (occupied sp) \timex (unoccupied spd) are included.
But correct EPS*.nolfc.d is rather from epsPP_lmfh script.

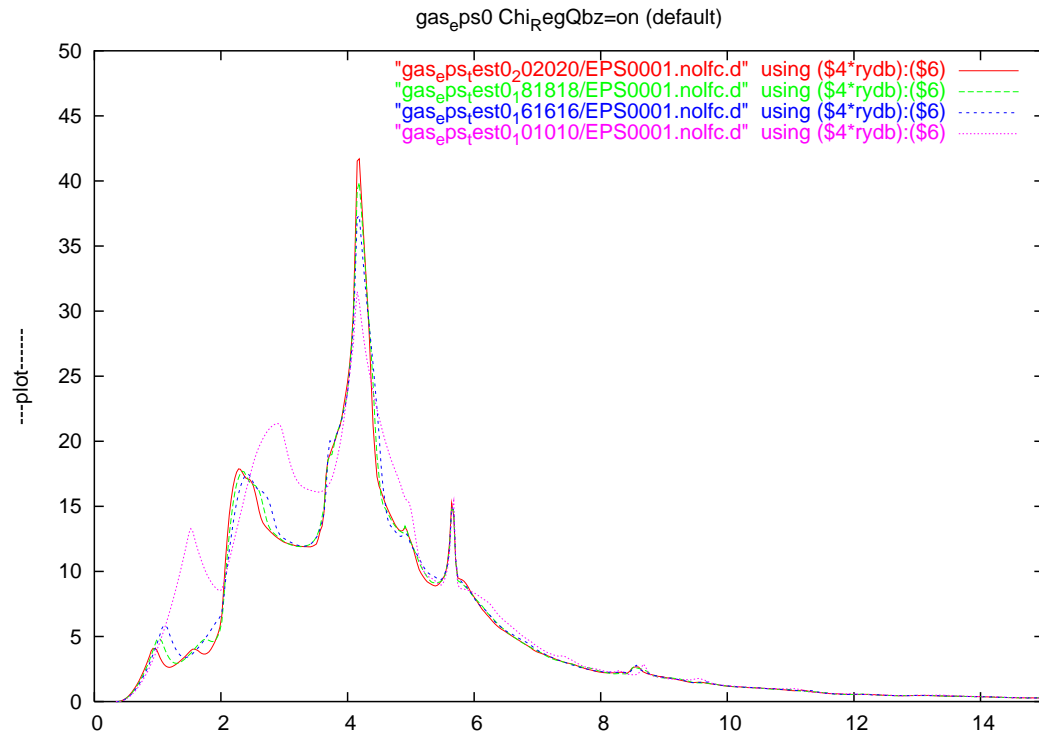


fig001

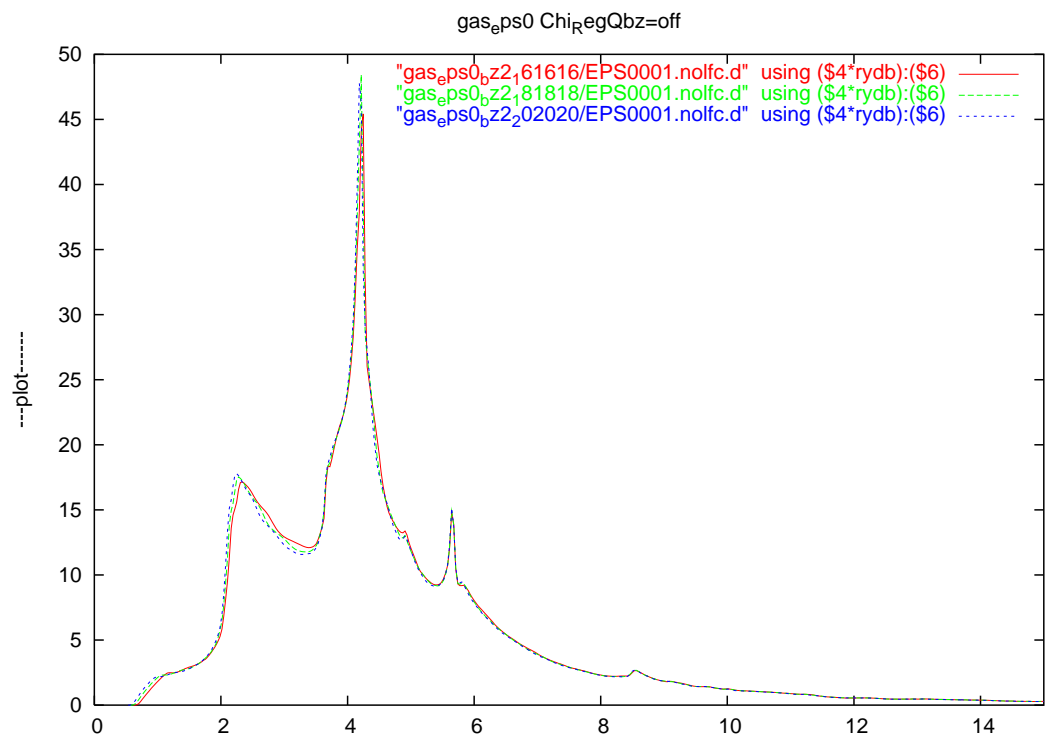


fig002

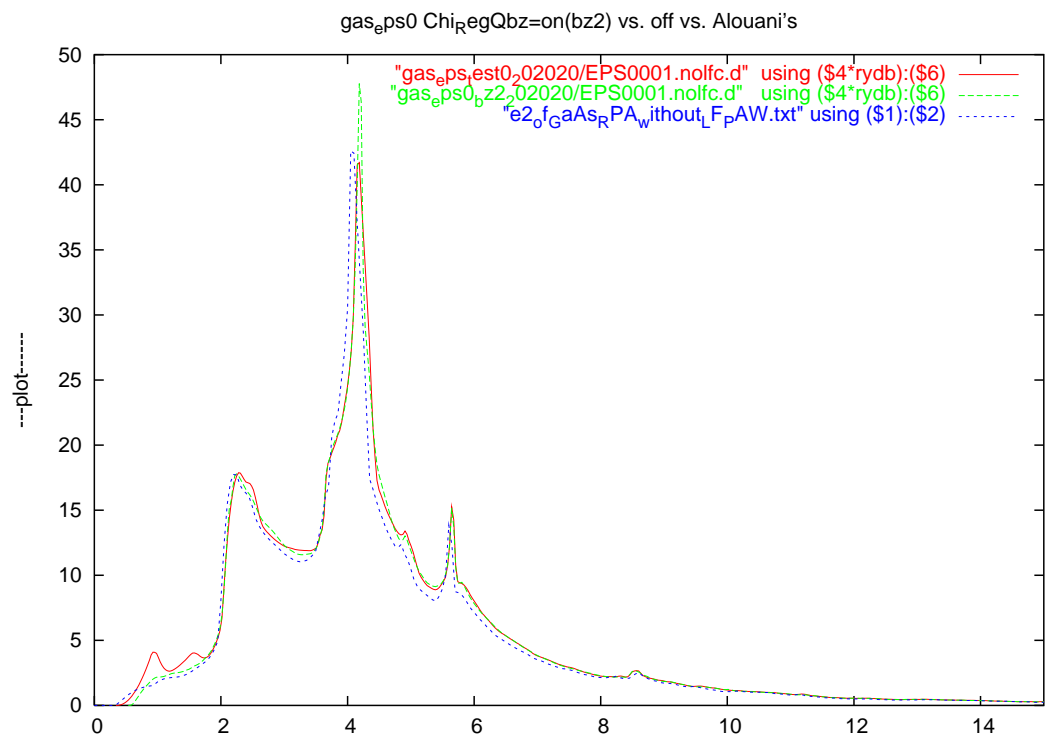


fig003

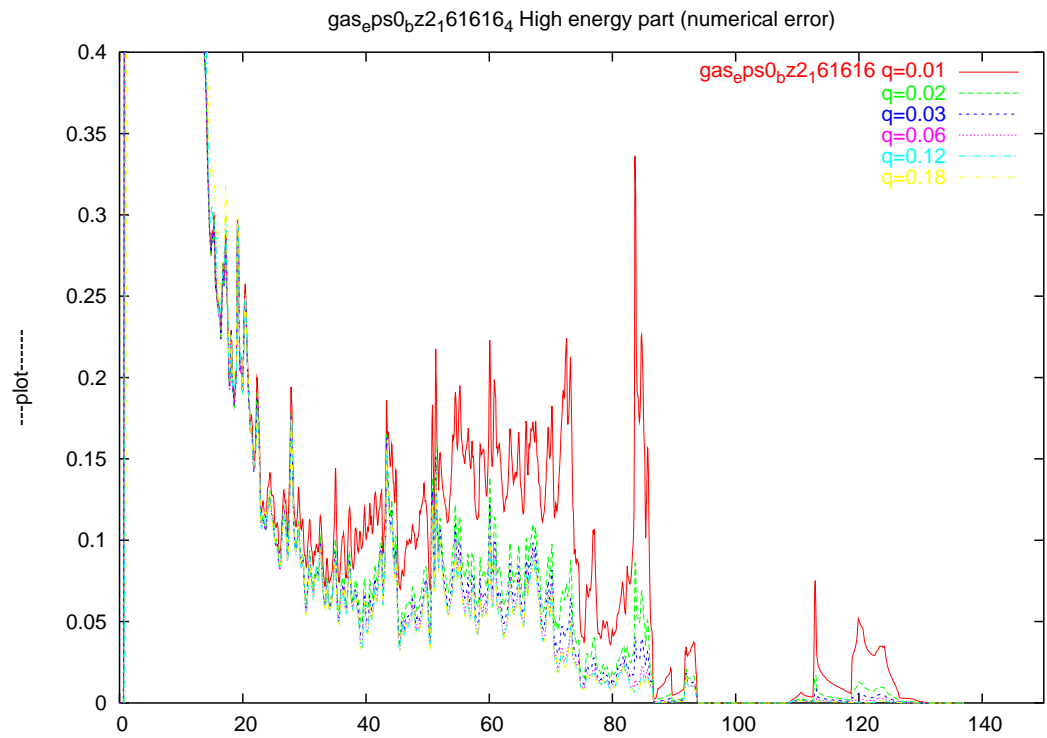


fig004

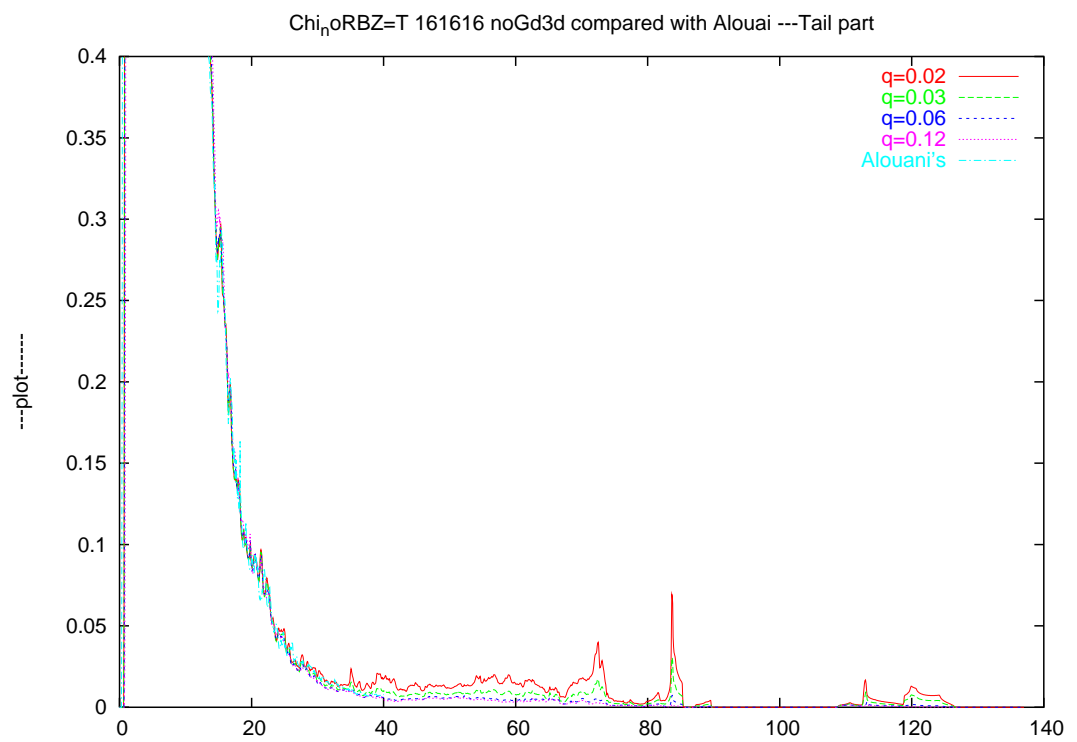


fig005

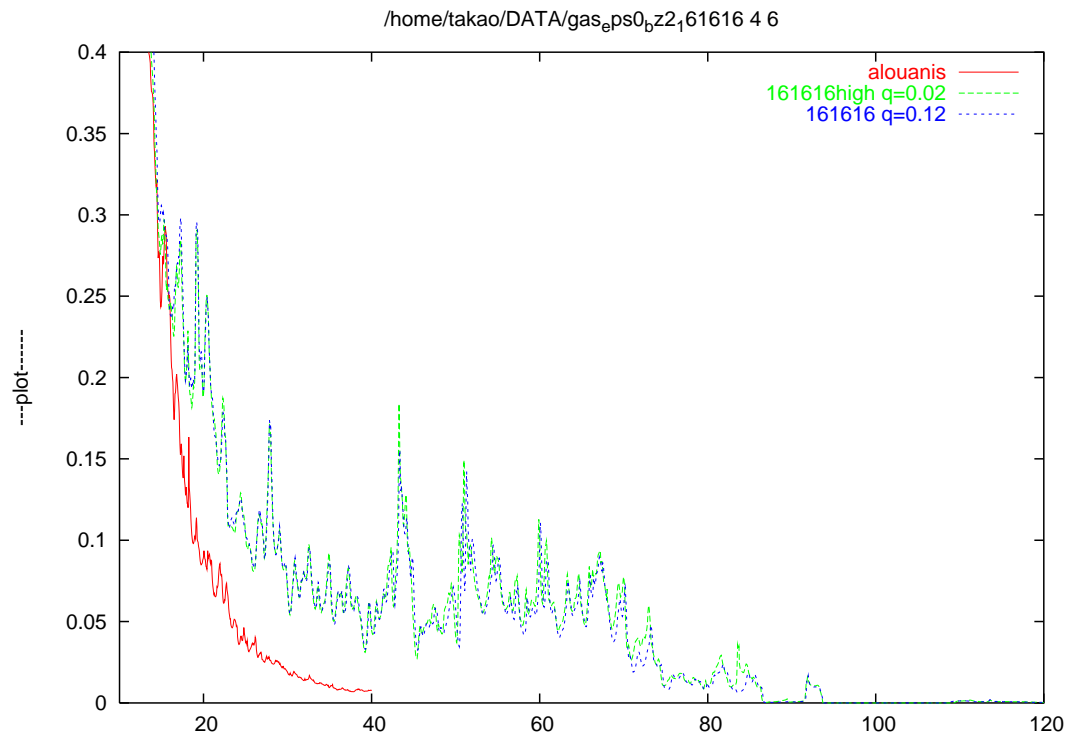


fig007

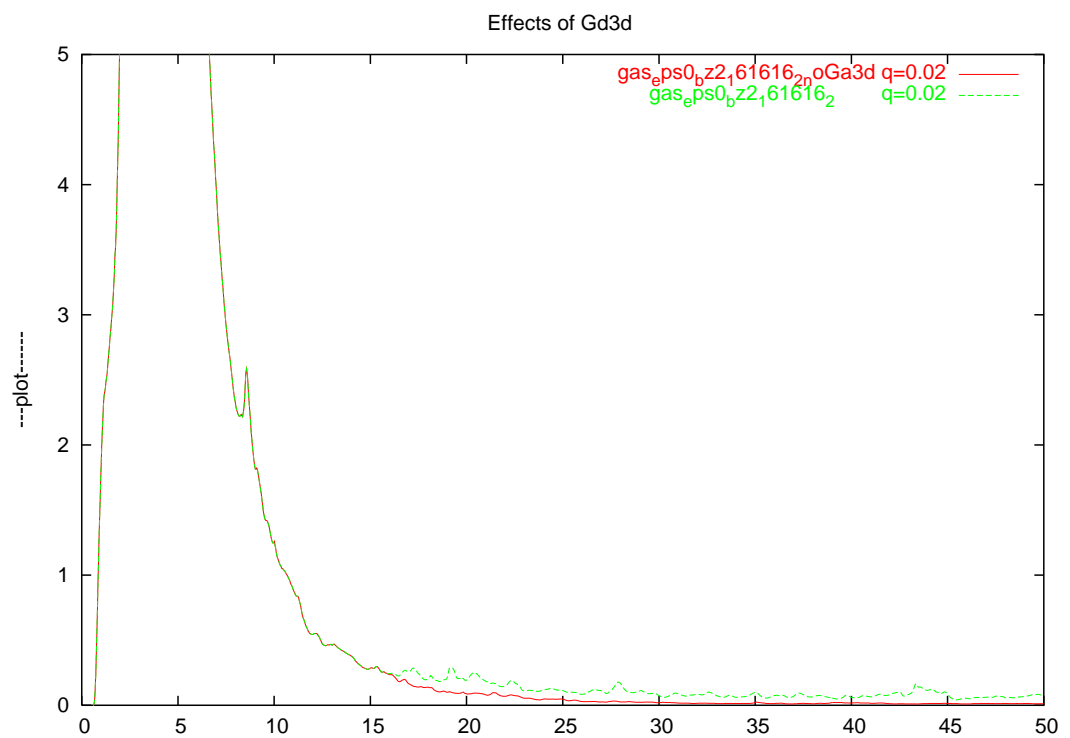


fig008

15 Used files

15.1 @MNLA_CPHI

m n l ibas

0	1	0	1	1	1
0	2	0	1	2	2
-1	1	1	1	3	3
0	1	1	1	4	3
1	1	1	1	5	3
-1	2	1	1	6	4
0	2	1	1	7	4
1	2	1	1	8	4
-2	1	2	1	9	5
-1	1	2	1	10	5
0	1	2	1	11	5

m is a magnetic quantum number, n is the degree of freedom which means 1 : ϕ , 2 : $\dot{\phi}$, and 3 : local orbital. l is the orbital angular quantum number. The match of orbitals and m number is shown by **job_pdos** command. The following number is the number of atom. And the next is the numerating number. It corresponds to the number in GWinput which is the most left one in the initial conditions.

15.2 CPHI and GEIG

We expand the eigenfunctions as the sum of the augmentation parts in MTs and the PW parts in the interstitial region:

$$\Psi_{\mathbf{k}n} = \sum_{\mathbf{R}u} \alpha_{\mathbf{R}u}^{\mathbf{k}n} \phi_{\mathbf{R}u}^{\mathbf{k}}(\mathbf{r}) + \sum_{\mathbf{G}} \beta_{\mathbf{G}}^{\mathbf{k}n} P_{\mathbf{G}}^{\mathbf{k}}(\mathbf{r})$$

Here, **CPHI** contains the information of $\alpha_{\mathbf{R}u}^{\mathbf{k}n}$ and **GEIG** contains $\beta_{\mathbf{G}}^{\mathbf{k}n}$.

References

- [Kotani et al.(2015)] Kotani, Kino, and Akai] T. Kotani, H. Kino, and H. Akai, Journal of the Physical Society of Japan **84**, 034702 (2015).
- [Kotani(2014)] T. Kotani, J. Phys. Soc. Jpn. **83**, 094711 [11 Pages] (2014), WOS:000340822100029.
- [Bruneval and Gatti(2014)] F. Bruneval and M. Gatti, in First Principles Approaches to Spectroscopic Properties of Complex Materials, Vol. 347, edited by C. Di Valentin, S. Botti, and M. Cococcioni (Springer Berlin Heidelberg, Berlin, Heidelberg, 2014) pp. 99–135.
- [Kotani and van Schilfgaarde(2002)] T. Kotani and M. van Schilfgaarde, Solid State Communications **121**, 461 (2002).
- [Friedrich et al.(2012)] Friedrich, Betzinger, Schlipf, Blügel, and Schindlmayr] C. Friedrich, M. Betzinger, M. Schlipf, S. Blügel, and A. Schindlmayr, Journal of physics. Condensed matter : an Institute of Physics journal **24**, 293201 (2012).
- [Kotani and van Schilfgaarde(2010)] T. Kotani and M. van Schilfgaarde, Physical Review B **81**, 125117 (2010) [5 pages] (2010).
- [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).
- [Rath and Freeman(1975)] J. Rath and A. Freeman, Phys. Rev. B **11**, 2109 (1975).