

**Full Potential Local Orbital
Minimum Basis
Bandstructure Scheme**



User's Manual

by

Klaus Koepernik

July 6, 2022

Contents

List of Tables	VII
Nomenclature	IX
1 Introduction	1
1.1 Installation procedures	1
1.1.1 Main program	1
1.1.2 XFBP	3
1.1.3 XFPLO	6
1.1.4 Further information	7
1.2 Program structure	14
1.3 Input structure	16
2 Version control	17
2.1 Upgrading	17
2.2 Downgrading	17
3 Files	19
3.1 General	19
3.2 File class 1	20
3.3 File class 2	24
3.4 File class 3	29
3.5 Directories	30
4 Basis states	31
4.1 General definitions	31
4.1.1 Orbital structure	31
4.1.2 Orbital classification	32
4.1.3 Radial functions	33
4.1.4 Multi orbitals	34
4.1.5 Default basis	34
4.1.6 Parameters	35
4.1.7 Modifications	36
4.2 Basis setup	36
4.3 =.basdef	37
4.4 Basis modifications	38
4.4.1 In FEDIT	38
4.4.2 PYFPLO	41

5 Algorithms	42
5.1 SCF k -mesh	42
5.2 LSDA+U	43
5.3 Total energy	44
5.4 Hamilton matrix elements (three-center integrals)	44
5.5 XC-functionals	45
5.5.1 Becke-Johnson (BJ06) and modified Becke-Johnson (mBJ)	45
6 Wannier functions with FPLO	50
6.1 Intro	51
6.2 The FPLO WF module	53
6.2.1 Files	57
6.2.2 Keywords	59
6.2.3 Definition of real spherical harmonics	62
6.2.4 Full relativistic spin-part	63
6.2.5 Problems	64
6.2.6 Saving memory (and time)	64
6.3 Examples	65
6.3.1 Hexagonal, Graphene, MgB ₂ : sp^2	65
6.3.2 Hexagonal, Graphene, MgB ₂ : p_z orbitals	70
7 Grep data from output: GREPPFLO	73
7.1 Specific data	73
7.2 Categorized data	75
8 Adding band weights: FADDWEI	77
9 Graphical interface: XFPL0	81
9.1 Structure viewer	81
9.2 Fermi surface viewer	81
9.3 Molecular/individual band weights	82
9.4 Density mapper	82
9.5 Help	83
10 Plotting program: XFBP	84
11 Occupation matrix manipulator: DMATEDIT	85
12 Optics: FOPTICS	87
12.1 Optical functions	91
12.1.1 General	91
12.1.2 Intra-band (Drude)	91
12.1.3 Intra-band and constant interband ϵ_∞	92

13 The dHvA-Module: FPLO/FDHVA	94
13.1 First stage (iso surface)	94
13.1.1 Input options	98
13.1.2 Cache file	101
13.1.3 Work flow	102
13.2 Second stage (dHvA spectrum)	102
13.2.1 Input options	107
14 Topological insulators	110
15 Band Unfolding	111
15.1 Citations	111
15.1.1 Copy right note	111
15.2 Introduction	111
15.2.1 Original idea	111
15.2.2 The local orbital connection	112
15.2.3 Alternative considerations	116
15.2.4 Summary:	120
15.3 Perturbations	120
15.4 Brillouine zones	121
15.5 Caveats	121
15.6 User input/output	122
15.6.1 =.unfold	122
15.6.2 choosing k-points	122
15.6.3 Fermi surfaces	123
15.6.4 Examples	123
16 Automation, scripting, pipe-mode, PYFPLO	124
16.1 Rules	124
16.2 Syntax	125
16.3 How to set up automation	128
16.4 Advanced features	130
16.4.1 Initial polarization, inital spin split	130
16.5 Example	131
Appendix	140
A Summary of Changes	140
B FAQ	145
Bibliography	147
Index	148

List of Tables

3.1	Band weights files.	26
7.1	Module output prefixes	76
8.1	Mapping spherial harmonics to functions.	78
12.1	Optical functions	89

Nomenclature

`stdout` the unix standard output file/channel

`stderr` the unix standard error file/channel

`stdin` the unix standard input file/channel

BZ Brillouin zone

CPA coherent crystal approximation

DFT density functional theory

FSM fixed spin moment

GGA generalized gradient approximation

LO local orbital

LOI local orbital integrals

MuO multi orbital

PID the unix process ID

PO polarization orbital

VCA virtual crystal approximation

`xc` exchange and correlation

(L)DOS (local/projected) density of states

L(S)DA+U local (spin) density approximation + U-functional

L(S)DA local (spin) density approximation

BZ Brillouin zone

`qn` quantum number

THCI three-center integrals

TI Topological insulator

v^{cf} Confining potential

v^{cryst} Effective crystall potential

WF Wannier function



Chapter 1

Introduction

This document shall help to understand the FPLO package. We adopt the following notions. Text in typewriter style refers to unix commands, FEDIT options, file content and such things. Text in blue typewriter are names of files (e.g. `=.in`). The special symbols FPLO, FEDIT, ... are placeholders for the actual (version-related) full names of these programs (if applicable).

Example: You have installed the binary `fplo22.00-62` and related binaries. Then the unix command (example)

```
FEDIT -pipe < ./=.pipefile 2>./+log 1>/dev/null
```

means

```
fedit22.00-62 -pipe < ./=.pipefile 2>./+log 1>/dev/null
```

for your installation.

In order to avoid confusion, the installation process **discontinued** to create links with the generic names `fplo`, `fedit` and so on after version 4 or so. The new policy is to call the programs with their full names. Beware, that the full name may include a platform specifier and even a user defined build branch name to use multiple architectures with a common file system¹.

Hint: Every important output produced by the FPLO package contains the version number of the program, which produced the output.

External materials, mostly additional information files and help screens from program parts are marked with a bar at the side, like for this paragraph. Help

1.1 Installation procedures

1.1.1 Main program

The distribution comes with the archive `FPL022.00-62.tar.gz`, the unpacking script `ftreeinst.sh` and the file `INSTALL`, which contains the following instruction:

¹There is an exception, the program `FPIOTEST`, which is used to manipulate input files on a basic level, usually from inside maintenance scripts. This program is still installed with a generic name, since we need the generic name in order for that scripts to work. And furthermore the program usually does not change in between versions.

```
#Starting with the downloaded tar-archive FPL022.00-62.tar.gz and the
#script ftreeinstall.sh, which both should now be in the same directory
#(where you want to install FPL0) execute the following commands

./ftreeinst.sh # now you must have an FPL0 directory

cd ./FPL0/FPL022.00-62/install

./MMakefile # answer all questions

cd .. # you are in FPL022.00-62 now

make
make # re-run make if errors occur (especially when option make -j N is used)
make

make install

# the main code is now installed in ./FPL0/bin
# containing:
# faddwei22.00-62-x86_64 grepfplo fpiotest22.00-62-x86_64 fdhva22.00-62-x86_64
# dirac22.00-62-x86_64 fplo22.00-62-x86_64 fedit22.00-62-x86_64 foptics22.00-62-x86_64
# fpiotest22.00-62-x86_64
#
# next we build the pyfplo package:
#

cd PYTHON

make # answer questions
#or for use with python3
make python3 # answer questions

# !!! Follow the instructions at the end of the make process to setup
# the python path !!!

cd .. # back in FPL022.00-62

# install XFBP

cd XFBP_rel

# read README and follow its instructions. Observe the python compilation mode
# setup decribed in README.

cd .. # back in FPL022.00-62
```

```
# install XFPL0

cd XFPL0_rel

# read README and follow its instructions

cd .. # back in FPL022.00-62
```

1.1.2 XFBP

The XFBP README contains:

```
-----
- QT setup
-----
```

QT5:

- * Install qt5.5.1 or later, including the development packages (header file, tools ...), including qt5help (or similar) if this is separate.
- * Or download qt5.5.1 or later and compile it.
- * Locate the qt5.5.1 qmake (or sometimes called with option: qmake -qt=5)
- * Check via

```
qmake --version
```

or

```
qmake -qt=5 --version
```

to find out.
- * chdir

```
cd XFBP
```
- * Edit QMAKE5 in makefile (not Makefile!) in order to call the proper qmake.
If 'qmake -qt=5' is qt5 then use

```
QMAKE5=qmake -qt=5
```

in makefile.
- * Edit QTVERSION in makefile as follows:

```
QTVERSION=5
```
- * go to PYTHON setup

QT4:

- * Install qt4.6 or later, including the development packages (header file, tools ...).
- * Or download qt4.6 or later and compile it.
- * Locate the qt4.6 qmake (or sometimes called qmake-qt4 or similar)
- * Check via

```
qmake --version
```

or

```
qmake -qt=4 --version
```

to find out.
- * chdir

```

    cd XFBP
* Edit QMAKE4 in makefile (not Makefile!) in order to call the
proper qmake.
If 'qmake -qt=4' is qt4 then use
    QMAKE4=qmake -qt=4
in makefile.
* Edit QTVERSION in makefile as follows:
    QTVERSION=4
* go to PYTHON setup

```

```

-----
- PYTHON setup
-----

```

To compile with python support (strongly suggested since we use it in the context of documentations and tutorials):

- 1) install python development packages (Python.h and python-config is needed)
Version should be >python2.7..., python3 is preferred
- 2) edit makefile and set the following variables to your python's executable and config program

```

PYTHON=python3
PYCONFIG=python3-config

```

It could be named python-config, python2-config or python2.7-config or python3-config or similar.

For PYTHON-version >3.8 we needed to include the ldflag --embed.
This is done automatically in the line starting with

```

PYCONFIGEMBED=

```

If this does not work, or python changes it's setup again corrections might be needed here.

Finally

```

make -f makefile clean
make -f makefile
make -f makefile install

```

OLD: to compile without python support:

```
make -f makefile clean
make -f makefile withoutpython
make -f makefile install
```

Beware that this is a development version, so some stuff does not work properly yet or does not get saved yet.

Call one of the following

```
xfbp +band...
xfbp +bweights...
xfbp file.xfp
xfbp -p paramfile
xfbp -p pyxfbpscript.py
xfbp -p pyxfbpscript.xpy
xfbp pyxfbpscript.xpy
cat "." | xfbp -p - # only for native script
```

or specialized

```
xfbp -xny datafile
xfbp -band bandfile
xfbp -bandweight bandfile
```

```
=====
= DEVELOPERS ONLY =
=====
```

For developpers. To remake the python binding from the src/parser/*.hpy install python2 ply (python lex and yacc), unfortunately it is not python3 yet.

then for compilation use

```
make c2py # python2
make clean
make
make install
# or
make c2py3 # python3
make clean
make
make install
```

To remake help: install sphinx, then

```
touch src/parser/*.hpy
make c2py3
make c2py
make PYHELP
```

1.1.3 XFPLO

The XFPLO README contains:

Install qt5.5.1 or a later version 5, including the development packages (header file, tools ...), including qt5help (might be in libqt5help5 on debian or in qttools5-dev/qttools5-dev-tools) or download qt5.5.1 or a later version 5 and compile it.

Locate the qt5 qmake (or sometimes called with option: qmake -qt=5), then

```
cd XFPLO
```

edit QMAKE5 in makefile (not Makefile!) in order to call the proper qmake

then execute

```
make -f makefile clean
make -f makefile
```

install into \$HOME/FPLO/bin via

```
make -f makefile install
```

Beware that this is a program under development, so some stuff does not work properly yet or does not get saved yet.

For help try

```
xfplo -h
```

Call one of the following for structure

```
xfplo -str
xfplo =.in
xfplo file.cif
```

For loading Wannier functions (or grid output functions)

```
xfplo =.in wfdata001
```

```
xfplo =.in grid_dens...
```

or one of the following for fermi surfaces

```
xfplo -fs
xfplo =.xef
```

The file versions are helpful if the files already exist.

```
#####
# install with QT4
#####
```

Install qt4.6 or later, including the development packages (header file)
or download qt4.6 or later and compile it.

locate the qt4.6 qmake (or sometimes called qmake-qt4)

```
cd XFPL0
```

edit QMAKE4 in makefile (not Makefile!) in order to call the proper qmake

execute

```
make -f makefile clean
make -f makefile all4
```

install into \$HOME/FPL0/bin via

```
make -f makefile install4
```

1.1.4 Further information

For further reference a copy of your-installation-path/FPL0/FPL022.00.../install/README contains:

```
Installation of FPL0-22.00-62
Ulrike Nitzsche (u.nitzsche@ifw-dresden.de)
Nov. 2013
```

I.Prerequisites

```
=====
```

What do you need to get FPL0-22.00-62 running?

At least a Unix system. FPL0-22.00-62 is tested under Linux on i386, x86_64
and ia64 architecture with the ifort compiler and it is planned to port
it to the gfortran compiler. Therefore the code should run (in near future)

on most Unix platforms with gfortran and gcc. (Test show that gfortran is not as runtime efficient as ifort.) It is not planned to port FPL0-22.00-62 to any Windows flavour.

The sources of FPL0-22.00-62 are a mix of C, C++ and F90 code. Therefore you need an ANSI conform C/C++-compiler and an F90-compiler. For commercial Unix systems (AIX, HPUX, Solaris) we suggest to use the native compilers to get best performance. Please take care to use latest versions and bug fixes. Unfortunately we are not able to test all the platforms and compiler versions. Therefore we can only support a gfortran/gcc or ifort/gcc installation there.

For all Linux (even for x86_64 Opteron) systems we recommend to use gcc and ifort10 (ifort is the Intel Fortran compiler, after a registration procedure for academic use you can get a unsupported non-commercial version from <http://www.intel.com/support/performance/tools/fortran/linux/> (this link might have changed)).

For the graphical software coming with FPL0 you need to install the qt libraries including development libraries. You need Qt version ≥ 4.6 but < 5 in the moment. Ports to Qt5 will happen once qt5 is becoming the standard.

There are some useful add-ons to FPL0-22.00-62. To use them you might need perl5-Interpreter (for developing the code and some auxillary tools)

To unpack the source tree you need tar and gzip. To use the installation scripts 'make' is required. On HPUX, we recommend to use gmake (gnu-make) instead of HP's own make (for a detailed explanation see <http://www.ifw-dresden.de/FPL0/faq.htm>).

II. Get the source (TODO)

=====

To get the source follow the instructions at <http://www.ifw-dresden.de/FPL0/conditions.htm>. After signing the license conditions and transferring the license fee you will get the source of FPL0-22.00-62 as MIME-attachment FPL022.00-62.tar.gz, another file ftreeinst.sh, and this README by email. Save these 3 files to disk and move them to your install directory. Let's say, you want to install FPL0 in your home directory. Hence do:

```
mv FPL022.00-62.tar.gz ftreeinst.sh README $HOME
```

Change into your install directory (in our example the home directory):

```
cd $HOME
```

Now we can start the installation procedure.

Starting with FPL0-3 we prepared our installation scripts for parallel installation of several FPL0-versions (including subversions) side by side. Therefor all FPL0-versions are located in a directory FPL0 in subdirectories FPL0-<Version>. In the directories FPL0/FPL0-<Version> you can view the source code, change the code and compile it. The binaries are installed into FPL0/bin as fplo-<version>, fedit-<version> and so on. To create the necessary directory tree, you have to start the installation with

```
sh ./ftreeinst.sh
```

If you don't have a directory called FPL0 no problems are expected, if there is a directory with this name you are asked to rename it.

Congratulations! The first step of the installation procedure is finished.

Now you will find a directory FPL022.00-62 in your FPL0 directory. Change to this directory:

```
cd FPL0/FPL022.00-62
```

Here, you find a directory "install". Change to it:

```
cd install
```

Now, you have to choose the type of installation:

If you use Linux with gcc and ifort or gfortran with default options go to III. Simple Installation on Linux with ifort as F90-Compilers .

To install xfplo and xfbp please go to

```
cd XFPL0_rel
```

and follow the instructions in the README file.
And

```
cd XFBP_rel
```

and follow the instructions in the README file.

To install pyfplo please go to

```
cd PYTHON
```

and type

```
make
```

or if wanted type

```
make python3
```

Setup the python path as announced at the end of make. Also consult DOC/pyfplo/pyfplo.pdf.

For installation of multiple builds of the same FPLO version go to IIIa.

In all other cases go to IV. Advanced Installation .

III. Simple Installation on Linux with ifort or gfortran as F90-Compilers

```
=====
```

You are in the directory FPL022.00-62/install .

To create the appropriate Makefile for your architecture, type:

```
sh ./MMakefile
```

You will be asked if f90 and gcc/g++ are your favorite compilers.

Answer the first question with either ifort or gfortran and the second/third question with [enter].

If ifort is used it comes with the MKL library. The next question is if you want to use the MKL-librarie's eigenvalue solver. Try answering y[es].

The installation tries to find the mkl path.

If it does not find it you will have to setup the linkage yourself or restart MMakefile and answer n[o], which results in using FPLO's inbuild eigenvalue solver.

If you want to setup the linkage yourself:

the MODULES/Makefile contains MKLLDFLAGS if MKL was requested by answering yes as explained above. Find out your MKL installation and edit this variable accordingly. Note: on 64-bit systems there are sometimes two version of this library. One has 32-bit integer variable and one has 64-bit integer variables. In case you use the library, with 32-bit integer arguments you need to make sure that the MODULES/Makefile contains the line

```
USE_MKL_DEFINE=-DUSE_MKL_LAPACK -I$(MKLROOT)/include
```

if the 64-bit integer argument version is used this line should read

```
USE_MKL_DEFINE=-DUSE_MKL_LAPACK_64 -I$(MKLROOT)/include
```

You can find this out by consulting the MKL library's documentation.

Intel has an online app to determine the proper library link settings.

After all questions are answered there will be a lot of system specific output, but you do not need to bother about it unless you suspect trouble.

Now, change into the FPL022.00-62 directory. Start the compilation procedure and install the executables in FPL0/bin:

```
cd ..
make
make install
```

Under certain circumstances it can be necessary to clean up the source tree first (e.g., if you compiled FPL0-22.00-62 previously on another architecture accessing the same file system):

```
make clean
make
make install
```

Now FPL0-22.00-62 is ready to work. To simplify its use, add the directory where your executables reside to your PATH :

for sh users:

```
PATH=$HOME/FPL0/bin:$PATH ; export PATH
```

for csh users:

```
setenv PATH $HOME/FPL0/bin:$PATH
```

To make this permanent for the next login, you have to add this line to your .profile or .login.csh respectively. For bash .bashrc and for tcsh .tcshrc should be the correct places. For local specifics you should ask your local system administrator.

Now you can read the Getting Started to learn how to perform calculations.

If you want to play with compiler options go to IV. Advanced Installation.

IIIIa. Simple Installation on heterogenous Linux clusters

=====
If you need several compilates for different machines on the same cluster do the following.

Go into the directory FPL022.00-62/install.

To create the appropriate Makefile for your architecture, type:

```
sh ./MMakefile 1
```

enter the build-branch name, when asked (e.g. machine name) and answer the next question with yes if the executables shall have the branch name appended at their end.

Now go one directory down and make the specific stuff

Note, that {branch-name} is a place holder for the actual name you gave during the MMakefile invocation.

```
cd ..
make -f Makefile.{branch-name}
make -f Makefile.{branch-name} install
```

This will make compilations in obj_{branch-name} sub directories.

When cleaning is needed use

```
make -f Makefile.{branch-name} clean
make -f Makefile.{branch-name}
make -f Makefile.{branch-name} install
```

IV. Advanced Installation

=====

We tried to create a rather flexible install script. The script MMakefile in FPL022.00-62/install is controlled by a configuration script located in FPL022.00-62/install/conf. It is called either <CC>-<F90>-<OS>-[RELEASE] or <CC>-<F90>-hostname.

```
<CC> is the name of the C-Compiler which you want to use
<F90> is the name of the F90-Compiler which you want to use
<OS> ist the name of the operating system on which
    MMakefile is running (you may get this name by typing uname -s)
[RELEASE] is the (optional) release number of the operating system
    on which MMakefile is running (type uname -r)
hostname is the hostname of the host on which MMakefile
    is running (type uname -n)
```

In these configuration files, architecture specifics are contained, concerning paths, libraries as well as similar things and compiler flags needed to compile FPL0-22.00-62.

MMakefile is looking for a configuration file in the following sequence:

```
<CC>-<F90>-hostname  
<CC>-<F90>-<OS>-[RELEASE]  
<CC>-<F90>-<OS>
```

Now we have to distinguish several cases:

You use a combination of architecture and compilers which is tested and supported by us, but want to take other compile options: Go to

1. Other compile options

You use an architecture which is supported by us but you want to take other compilers: Go to 2. Other compiler

You want to use an unsupported architecture: Go to 3. Other architecture

1. Other compile options

Let us assume you want to compile FPL0-22.00-62 on a 32bit-system running Linux with ifort but options different from the default options.

The name of your host is, e.g., MyHost. Then you have to go into the configuration directory and copy the appropriate configuration file gcc-ifort-Linux-i386 to gcc-ifort-MyHost. After that you can edit the lines concerning the compile options (CFLAGS, CFEDITFLAGS, F90FLAGS, SPECIALF90FLAGS), run MMakefile in FPL022.00-62/install and compile in FPL022.00-62 .

```
cd FPL022.00-62/install/conf  
cp gcc-ifort-Linux-i386 gcc-ifort-MyHost  
vi cc-f90-MyHost  
cd ..  
sh ./MMakefile  
cd ..  
make clean  
make  
make install
```

2. Other compiler

Let us assume you want to compile FPL0-22.00-62 on a 32bit-system running Linux with gcc but different F90-Compiler. The name of the new F90-Compiler is MyF90, your host is e.g. MyHost. Then you have to go into the configuration directory and copy the appropriate configuration file gcc-ifort-Linux-i386 to gcc-MyF90-MyHost. After that you can edit the lines concerning the compilers (F90, F90FLAGS, SPECIALF90FLAGS), run MMakefile in FPL022.00-62/install and compile in FPL022.00-62. Be careful with the compile options, at least you have to switch on ansi behaviour. Unfortunately, it is impossible to give more hints for choosing appropriate compile options.

```
cd FPL022.00-62/install/conf
```

```

cp cc-f90-OSF1-V5.0 cc-MyF90-MyHost
vi cc-MyF90-MyHost
cd ..
sh ./MMakefile
Important: Answer the question about F90-Compiler with MyF90 !
cd ..
make clean
make
make install

```

3. Other architecture

This is the most difficult case, and we can give you only some general hints:

Choose a configuration file that seems to stem from a similar architecture, copy it on a file with the known naming conventions, than read every line carefully. Some are self-explanatory, for others you will need hints about your operating system.

Please read FPL022.00-62/install/README.advanced for more information. Feel free to contact us in difficult cases.

V. Miscellaneous

=====

You can change the names of the executables to be created in the following files:

```

FPL022.00-62/MODULES/Makefile.src (fplo, dirac)
FPL022.00-62/BP/Makefile.src (bandplot)
FPL022.00-62/FEDIT/Makefile.src (fedit)

```

You can change the directory where the executables reside in the following file:

```

FPL022.00-62/install/MMakefile (change the line: installdir=$srcpath/bin )

```

In the case you changed the executable's names and/or the install directory, please repeat the install procedure, in order to let the changes taking place.

For bugs in this document please send an email to fplo@ifw-dresden.de.

1.2 Program structure

The FPLO package consists of the source tree, which you hopefully compiled succesfully, and of the executables, which reside in `$HOME/FPLO/bin` in the standard installation. See installation instructions in `FPLO.../install/README...` for more information. There are a couple of binaries and a number of scripts. The binaries are the following:

FPLO	The bandstructure solver for the Kohn-Sham problem of bulk systems and molecules. It is a monolithic program, which performs the whole self consistent calculation. There are no such things as a bundle of standalone programs, which handle subtasks of the whole problem. The input of FPLO is handled via the input editor.
FEDIT	The input editor. It handles the input editing for FPLO, DIRAC and BANDPLOT. There is basic help available via <p style="text-align: center;">FEDIT -h</p> <p>Furthermore, there are help screens for every menu, which explain a number of aspects. Please read them. FEDIT has a pipe-mode, which is designed for automatic input management. It allows manipulation of input files by scripts, without messing up the input structure.</p>
XFPLO	Display and manipulate structures, display fermi surfaces, define path through BZ for band plotting. Sec. 9
XFBP	(X11-FPLO-BandPlot) Plot bands and other functions, this program works a lot like xmgrace. Sec. 10
DMATEDIT	Edit LSDA+U occupation matrices for setting up starting configurations to enforce possible metastable solutions. 11
FADDWEI	Add/manipulate weights in band weight files. Sec. 8
FOPTICS	Manipulate the optics output to generate other optics functions from +imeps . See Sec. 12
FDHVA	Uses the files +iso_b... as input to calculate the dHvA spectrum (area in Tesla versus field direction). See Sec. 13
GREFPLO	extract information from output files. 7
BANDPLOT	Old, since version 14 XFBP provides better services. The bandstructure plotting utility. It is normally used by FEDIT (and not called from the command line) to create band-structure and band-weight plots. Call <p style="text-align: center;">FEDIT -bandplot</p>
DIRAC	The standalone atom solver (spherical atoms). This solves the (non-) relativistic DFT problem for spherical atoms. To edit the input, call <p style="text-align: center;">FEDIT -atom</p> <p>To learn more read the help screens in this mode. (SIC is not yet working!)</p>
FPIOTEST	A utility for input manipulations. This is mainly used by the scripts of the distribution.

1.3 Input structure

The main input files use a special syntax, which has some aspects in common with the C-programming language. It is, however, not one of the common data languages around, but a private one. The reason to introduce this feature was the general design of the input handling and the aspired independence from external libraries, which can not be assumed to be present on every machine. The package contains an input parser, which is accessible to the C and F90 code. This reduces the need of FORTRAN IO management and thus increases the flexibility of version management and such things.

As a consequence, the user normally cannot and **should not** alter the content of the input files. **All input settings are meant to be changed with the help of FEDIT only.** (Exceptions are manipulations, which are done by XFPLO.) In fact FEDIT is very easy to handle and there is no need for manual input file manipulations. (Some scripts also change the input, using FPIOTEST, which in turn uses the mentioned parser to achieve this goal.) For batch jobs/scripting/automation there is a special mode called pipe-mode (Chap. 124) in FEDIT. The new pyfplo python package allows input manipulation as well and it's use is strongly encouraged (see [../pyfplo/pyfplo.pdf](#)).

The input, which is needed by the various executables is managed in a particular way. The executables create communication files (`+fedit`, `+fedithelp`), which tell FEDIT how to process input and how to manage menus. FEDIT uses some methods of back-communication² to the executables to have them responding in certain ways. This kind of communication is designed to avoid the user to edit input files directly (with a few exceptions).

If FPLO is called in an empty directory, it will immediately terminate with a message telling that it created one of those communication files. On the next invocation it creates standard input files and again terminates. All this is not seen by the user, if he uses FEDIT.

Important: To save a calculation it is sufficient to save all files of class 1 with the prefix '=' (see Chapter 3). If one uses such an archived calculation later to, say, create some additional output data, the first call to FPLO would stop as described above (unless FEDIT was used before). So, don't be afraid if the code exits, stating that it created the communication files. Just restart it and everything will be fine.

New since version 17: all these tasks have been moved from FPLO to internal code in FEDIT. This means that we no longer have this communication between the two codes. FEDIT is now tightly bound to the input management of it's version, except in DIRAC mode, where we still use the old scheme described above.

²These are command line options (BANDPLOT), status variables in the input files (`=.dirac`) and deletion of certain files.



Chapter 2

Version control

The package uses rather strict version control rules. A version number has the form “`x.xx`”, where `x` is a placeholder for a digit. Furthermore there is a release number, which has the form “`x`”. A full version-release number is the version number followed by a “`-`” followed by the release number like in `22.00-62`, which means version 22.00 release 62. Additionally, a string is attached to the executable names in order to differentiate between different architectures, e.g. “`-i386`” or “`-x86_64`”. Optionally, the user can chose to add a specific name at the end during the installation process, which is usefull on heterogenous clusters, where different compilations for different architectures are needed. This is achieved by adding the option “`1`” to the call: “`install/MMakefile 1`” during installation. Read the `install/README` (Sec. 1.1) file.

Every major input file contains its own version number. Every executable has its own internal version number. To avoid problems, one usually cannot use files with programs of different version.

There is a simple rule: **The version number of the code is changed whenever input has changed.** If only the release number has changed the input is not altered. Normally, changes of the input consist of adding something. In rare cases, the structure changes. For this reason it is not recommended to manipulate version numbers of files by hand.

2.1 Upgrading

If the version number of FPLO has been increased it is rather simple to upgrade the old files. FPLO upgrades the files in the working directory automatically if they have older version numbers. Just call FEDIT or FPLO and have a look at the output. If FEDIT is used it will ask the user before performing the upgrade, while FPLO will do it without asking!

Attention: We strongly recomment to copy the whole directory (assumed you organize different calculations in different directories) before executing the upgrade. The reason is that the numerical results between different versions may slightly differ due to numerical changes/improvements¹.

2.2 Downgrading

Sometimes, it is nessecary to downgrad files to an older version number, mostly to undo erroneous upgrading of files belonging to a series of calculations, which is intended to be completed with the older FPLO version. However, sometimes one may wish to recalculate something with an older version for comparison (although this should be a very rare case for the normal user).

¹After all FPLO (like any other code) solves the problem approximately, although rather accurate.

Downgrading means that some information gets lost. Thus, you should consider to **save a backup copy** of the files **before** you **downgrade**. Have in mind that some calculation-modes are not available in older versions of FPLO. Downgrading of calculations using such modes makes no sense.

Downgrading is a bit more complicated than upgrading. There is a `perl` script in the distribution, which handles the complicated restructuring.

1. Execute `fdowngrad.pl` at the command line and answer a few questions. As a result the input files are downgraded. The old files are copied to a backup directory called `fdowngrad_backup[n]`, where `[n]` is a number which is increased on every call to the script.
2. Next call the older version of FEDIT belonging to the downgrad-version to regularize the files. Go to the `symmetry` submenu and use `update`. **NEVER SKIP THIS POINT!**

Attention: Downgrading from version ≥ 9.07 to version < 9.07 includes a subtle step, which will be explained here in detail. The density file information has been enriched in several versions. If one would use the file as it is with an older FPLO version, the density would be interpreted in the wrong way^a. To avoid this it is necessary to map the new density onto the old file format, which can only be done with an FPLO version belonging to the density file (≥ 9.07). The necessary steps are

1. Edit the file `=.densconvert`: put in the main version number (like 9 or 7).
2. Run the FPLO corresponding to the density file. It will stop after the conversion.
3. Delete `=.densconvert`, otherwise the program might get confused.

Important: `fdowngrad.pl` tries to do these steps for you. This will only work if you have the proper FPLO version, ideally the one which has the version number of the files to be downgraded. If everything fails, you still have the backup. In the worst case you need to re-iterate the density^b.

^aThis basically results in the destruction of the (already converged) density.

^b:-)

Pitfalls: Some information from files of newer versions may be invalid in older version executables. This is seen if the older version FEDIT is used on the downgraded files (at least on exit, there will be a message.). Correct the input and rethink if you really did what you wanted to do.

Example:

Downgrading a full relativistic input say version 4.00 to version 3.00 (where full relativistic mode is not available) will leave the relativistic mode-data untouched in `=.in`. But, this is invalid input in `fplo3.00-6`. Calling `fedit3.00-6` and executing `quit/save` will result in an error message about an invalid value of the variable `relativistic`. Go to the `relativistic-select` box and select a proper option. Now, you can `quit/save` and you have valid input.



Chapter 3

Files

3.1 General

The files used by FPLO are classified into 3 major classes.

1. Files, which contain input or both input and output data and which are necessary for a successful restart of a previously converged calculation. These have the prefix `'=.'`. One should not delete these files nor use them for different calculations with different parameter settings. A safe rule is: each calculation is done in a separate directory. One may copy all these files into a new directory, modify the input with FEDIT and start a new calculation. This is especially useful in the case of slight changes of some parameters, in which case the density of the previous calculation is usually a better starting point than the atomic density (created in the very beginning of a calculation if `=.dens` is absent). If, however, the number of sites in the unit cell or the type of elements is changing, an initial density is needed (see FPLO output).
2. Files, which are mainly output files and which are not necessary for a successful restart of a previously converged calculation. These files have the prefix `'+'`. In general one can delete these files (`rm ./+*`) after successful calculations.

Caution: the bandstructure is written into `+band...` or `+bweights...`. These files are used by XFBP (BANDPLOT) to create a picture of the bandstructure. So, if one inadvertently deletes these files one needs a single step calculation, starting from a converged density, to recreate them.

Furthermore, there are files, which are output of FPLO but input to other programs or used for caching data, e.g. `+imeps`, `+isoergcache...` and `+iso_b...`

Important: Many (new) unix tools will interpret the `'+'` sign as an option flag. To use these tools with `'+'`-files, specify `./+file` instead `+file` on command line¹. Example:

```
less ./+band
```

3. Files, which are either pure output or status memory files for graphical tools. These files have no prefix. Now, the grid output data and Wannier function data also have no prefix.

Example: `wfdata...`, `grid_...`, `.net`-files, `.ini`-files

In the following the most important files are described in more detail.

Most input files are copied to the output. The script `fout2in` can be used to extract input files from any output file.

¹When FPLO was designed, this habit of unix (Linux) tools was not yet widespread, and hence not recognized by the FPLO authors.

3.2 File class 1

The prefix '=' indicates their primary use as input to FPLO. Nevertheless, they are partially output files. FPLO uses the following files²:

`=.in` is handled by FEDIT, it is **no recommended** to manipulate them manually

`=.sym`

Obsolete since version 17. It contains the crystal symmetry. From the information contained in this file `=.in` (and before version 6 `=.basis`) are recalculated. Recalculation happens if `=.in` is absent or if the status flag in `=.sym` requires an update. Normally this is done by FEDIT. On update, the non-default settings of the existing `=.in` will be retained unless the symmetry in `=.sym` contradicts these settings, in which case the default settings are used. The code will notify these reset-events after update³.

In a standard FPLO calculation run (no update action) the symmetry settings of `=.in` are used even if `=.sym` contains a different symmetry. Normally, if the update function of FEDIT was used, the symmetry settings of both files are equivalent.

If `=.in` exists and `=.sym` is absent, FPLO will extract a valid `=.sym` from `=.in` (this is usually done while invoking FEDIT).

`=.in` contains the major control data for FPLO, except for the the basis. You can manipulate it interactively with FEDIT or automatically with the FEDIT-pipe-mode. Please read FEDIT help screens! Some information can be modified via XFLO. Since version 17 the symmetry update which required the file `=.sym` is handled differently, which makes `=.sym` obsolete. The update functionality is still present though, which means that some data need resetting to default values when the symmetry settings change. FEDIT handles these things.

`=.dens` contains the density contributions of all sites (in terms of radial functions, which are the coefficients of the angular momentum expansion of the **overlapping** site densities). This file serves as input and output for FPLO. It is created by means of a simple atom-like calculation on FPLO startup, if not yet existing. The density file may be re-used in other calculations (often a better starting point than the atom densities) if the number of sites in the unit cell and the type of atoms are equivalent.

`=.dmat_init` contains the occupation number matrices for LSDA+U. This file's content is duplicated in `=.dens`, so `=.dmat_init` can get lost. However, if it is present it will overrule the information contained in `=.dens`. When FPLO runs it writes/updates this file with the current occupation number matrices. Its main purpose is to present the data in an easily editable format in order that one can manipulate the (initial) occupation number matrices to drive the calculation to one of the possible (local) LSDA+U minima. The format is quite unrestricted, however not fool-proof. So be careful when editing. In any case this file will be recreated in the next FPLO run. If you want a fresh copy just delete it. **New: DMATEDIT can be used to manipulate this file in order to set up starting conditions for (meta) stable solutions. This program allows to rotate the local axis in which the orbitals are defined. (Sec. 11)**

²There may be more class-1 files, which are not documented here. However, the normal user is not expected to need them. There are some utility programs for the package, which use the same file name convention and thus have additional class-1 files (e.g. BANDPLOT). These are also not documented here.

³When FEDIT is used, see the protocol screen after `symmtry update`.

=.kp If this file is found by FPLO, the bandstructure (written to **+band...** (see Page 26)) is calculated at the points given in **=.kp**. This is e.g. used to calculate Fermi surfaces with XFPLO (Sec. 9.2). The points are given in units $\frac{2\pi}{a_0}$, thus i.e. for bcc lattices the line Gamma-H consists of all points between (0,0,0) and (1,0,0).

Format:

line 1: number-of- k -points [only partially occupied bands] [lower offset] [upper offset]

line 2,...: one k -point per line (3 real numbers, separated by space)

Explanations:

The three entries behind the number of k -points in the first line are optional. They are used by the newer versions of the program XFPLO to reduce file size.

[only partially occupied bands] a logical (t/f). If this is t only partially occupied bands are written to the files **+band...** or **+bweights...**. This reduces file sizes considerably, especially when used in conjunction with the Fermi surface program XFPLO .

[lower offset] Additionally that number of bands below the lowest partially occupied band are written to the files.

[upper offset] Additionally that number of bands above the highest partially occupied band are written to the files.

=.basdef The default basis and optional modifications are used if this file is not present. This file is copied to the output like the other main input files. Hence, we can extract the (default) basis definition file from any output. For the newer treatment of the basis read 4. Formerly, **fout2in -b** could be used in order to extract the basis definition, used in the output file under consideration.

=.unfold Unfolding is explained in a separate document (Chap. 15). Create the input by hand or use the unfold editor of the structure mode of XFPLO. Sec. ??

=.addwei Add band weights. Used by FADDWEI.(See. 8)

=.bwdef Molecular/individual band weights can be defined via this file (Sec. 9.3). The advantage is that the resulting **+bweights...** file can be much smaller if the user only needs a few fatbands. Furthermore, molecular patterns help elucidate bonding behaviour. Call XFPLO **-bw** or XFPLO **=.bwdef** to edit the file. This file can be specified in the FEDIT bandplot menu after which FPLO will use the bandweight definitions from this file instead of the default when creating fatbands.

=.densmap Map density files from one structure onto another structure. See Sec. 9.4.

=.xstr Saved by XFPLO . It contains all settings needed to replicate the structure, lighting, fog, polyhedra and so forth. See Sec. 9.1. This has nothing to do with the structure used by FPLO.

=.xef Saved by XFPLO . It contains all settings needed to replicate the Fermi surface plot. See Sec. 9.2.

=.xfp Saved by XFBP . It contains all settings needed to replicate the plot created by this program. See Sec. 10.

=.cmd Saved by XFBP . It contains a user written script used by this program. This enables automation. See Sec. 10.

=.coeff **Obsolete since version 10.00**, because it has been turned into the switch `Output +coeff` file in the FEDIT bandstructure submenu.

=.atcharge **Obsolete since version 10.00**, because input was moved to FEDIT charges menu. On version update the data is transferred to `=.in` (FEDIT).

Somebody may want to use the virtual crystal approximation (VCA), which consists basically of introducing non-integer nuclear charges. This may be done by defining the content of this file.

Format:

line 1: number of lines following

line 2,...: number of Wyckoff-position followed by the nuclear charge

The nuclear charge must not deviate from the nominal nuclear charge Z of the element by more than ± 1 . Since basis sets are element specific it might make a difference, whether one uses $Z - \delta$ or $Z + \delta$. The basis set is always chosen according to the nominal nuclear charge. However, basis parameters (and also other specific parameters) are interpolated between Z and $Z \pm \delta$.

=.mol_charge **Obsolete since version 10.00**, because input was moved to FEDIT charges menu. On version update the data is transferred to `=.in` (FEDIT).

In molecule mode this defines the all-over cluster/molecular charge. The file contains one real number (the charge) in the first line. A positive number means less electrons than neutral.

`=.basis` (obsolete since version 6.00) contains the basis definitions. This includes the definition, which orbitals enter the calculation (core/ semi-core/ valence/ polarization) and the initial definition of the compression parameters (x_0) of the confining potential. The file content may be manipulated using FEDIT.

In auto optimizing mode (option BASIS_OPTIMIZATION) the file is updated with the new value of x_0 while FPLO is running. So, be aware if FPLO is running and FEDIT is used meanwhile, that the file content may have changed on disk during the FEDIT session. You will be prompted to overwrite `=.basis` on exit (quit/save) in case that the content changed, while the FEDIT session was active. Usually it is correct **not** to overwrite the `=.basis` content in such a situation.

For all these files, read the help screens of FEDIT!

`=.densgrid` **Obsolete since version 9.00**, because input has been moved to a new FEDIT sub-menu.

To create real space density or potential plots create this file. It contains header informations and grid data. It may contain comments (line starting with '#') everywhere. It may contain empty lines.

Comments at the top of the file (before data lines) may contain control settings. All other lines contain real space vectors in cartesian coordinates. For each vector the density/potential is plotted into the file `+densgrid`.

Header:

Important: please **no** tab-characters, only space!

All header control data are optional. Below we give the complete list of possible controls and their options. Any combination and order of options is valid. Options must be separated by at least a single space. The colon after the control key-word (output,data,...) may be separated with spaces. Any of the control lines described in 1-3 may be omitted, which forces default behaviour.

1. # output: comments emptylines emptylines_with_space stop

comments	output all lines starting with any space followed by '#'. If '# output' is the first line, it controls the output of all lines following, including itself.
emptylines	copy all empty lines of <code>=.densgrid</code> into <code>+densgrid</code>
emptylines_with_space	put a line with one single space into <code>+densgrid</code> for every empty line of <code>=.densgrid</code> (some plotting programs need this to separate data sets)
stop	stop after plotting, please be aware that you need a converged calculation to get reasonable densities/potentials!

emptylines_with_space wins over emptylines, if both are specified!
2. # data : index point total spin spinup spindown

index	print index of grid point in first line
point	print grid point
total	print total density/potential
spin	print spin density (i.e. $n^\uparrow - n^\downarrow$)/xc-field
spinup	print majority density/potential
spindown	print minority density/potential
3. # type : density potential

```

density    plot density data
potential  plot total-potential data

```

The last `type` specified, will win.

Plotting is done at the beginning of each iteration cycle after the potential is calculated.

Default:

1. `type:` density
2. `output:` no comments, no empty lines no stop
3. `data:` point total spin spinup spindown

The order of output data is fixed. So the `data`-options may be given in any order, but the output in `+densgrid` allways has the order: index point total spin spinup spindown. Of course, some of the fields may be absent, if omitted in `'data: ...'`

Remark: If some grid point falls onto an atomic position, the onsite contribution of this atom is neglected, since relativistic densities diverge at the nucleus ($s^{1/2}$ and $p^{1/2}$ orbitals diverge). Same holds for the potential (relativistic or not).

`=.ldos` **Obsolete since version 4.01**, because input was moved to FEDIT `bandplot` menu

To create lm -resolved densities of states, create this file in the directory, where the calculation is done. If FPLO finds this file the local orbital-resolved DOS is calculated.

Format:

line 1: number of lines following

line 2,...: number of site, for which a resolved DOS should be created

'sites' means all sites in the cell, not the Wyckoff positions! The definition of sites is found in the FPLO-output section 'UNIT CELL CREATION', part 'Atom sites'.

3.3 File class 2

The prefix '+' indicates the primary use as output from FPLO. Nevertheless, the files are partially input files for subsequent runs or tests. (Examples: debug files like `+basis`, `+loi`)

`+dos.total` This file and all other DOS files (except the `+(i)ldos` files) are created if the option "CALC_DOS" is set or if "Bandstructure plot" in the `bandplot` menu is true unless the option "NO_DOS" is set.⁴ It contains the total density of states.

`+dos.total.l` contains the l -projection of the total density of states. E.g. the d -DOS is the sum of all d -orbital contributions of all atoms to the total DOS. See comments inside the files to see which angular momentum l is contained in the file. The numbers in the file suffix are just counters.

⁴In CPA calculations there is no "bandstructure plot" option. Instead the Bloch spectral density may be calculated.

+dos.sort contains the sort projected DOS. This is the sum of all DOS contributions of all sites belonging to the same Wyckoff position (sort).

+dos.sort.nl contains the sort and *nl*-projected DOS. This is the sum of all *nl*-DOS contributions of all sites belonging to the same Wyckoff position. See comments (lines starting with '#') in the file.

+idos... These are like the **+dos.**-files but with the integrated DOS. Integrated DOS is only created when the option `Plot IDOS` in the `bandplot` menu is set.

+(i)ldos.site.nl These files are created for all sites given in the `bandplot` menu entry 'Local DOS sites' (this entry is a space separated list of site numbers) (see page 24 for older input versions) and contain the site and *l,m*-resolved DOS. The file numbers **nl...** are running indices, the real *nlm* numbers are written in comment lines ('#') inside the files. In full relativistic mode also *ljμ* projected files get produced.

+imeps This file contains the inter-band part of the optical function $\text{Im}\varepsilon(\omega)$ it is used for optics (see Sec. 12).

+isoergcache, +isoergcache_wan These files contain the cached band energy (and band weights) from the iso surface stage run of the dHvA module. They serve as input for a speedup of a possible re-run of the iso surface stage. (see Sec. 13).

+iso_b..._p..._spin... These files contain the iso surfaces for the dHvA module. They are produced by the iso surface stage of the dHvA module and serve as input for FDHVA. (see Sec. 13). The numbers after **_b** denote the band index of the corresponding band, **_p** denotes the number of the part of the sheet (if the sheet for this band index has disconnected parts) and **_spin** denotes the spin. For full-relativistic calculations the spin suffix is missing, since spin is not a good quantum number in this case.

+area_vs_angle..., +mass_vs_angle... These files are output of the dHvA stage (FDHVA) of the dHvA module (see Sec. 13). They contain curves of extremal areas in *T_{eff}* vs. a continuous angle variable through the main field directions. The mass files contain the corresponding masses. They can be loaded using a prepared script file XFBP `area_vs_angle.cmd`.

+coeff If `coeff output` is switched on in the FEDIT bandstructure submenu this file is created and contains the wave function coefficients *C* of the reduced valence problem

$$HC = SCE$$

This file is usually very large. It can however be used in conjunction with `XFPLO -bw` to create hand tailored band weights.

Attention: The full wave function $\Psi = \Phi C$ is constructed from this information (C) **and** from the core and valence orbitals $\Phi_{c,v}$ and core-valence overlap S_{cv} , which are not contained in `+coeff`.

`+error` Created (and updated), while FPLO is running. It contains a summary of warnings and error messages. The update mechanism does not work on some platforms. Thus at the moment, it is best practice to check the FPLO output, since all messages are duplicated to standard output.

A lot of messages contain the number of the iteration step, where they occurred. Normally, only the messages of the last iteration step (before convergence) are relevant.

`+run` contains the hostname and the PID of the last run of FPLO. If it is still running, this information may be used to kill the job.

Caution: Killing FPLO, may cause loss of the `=.dens` file and therefore loss of the calculation result, in case that FPLO is just writing the file `=.dens`, when it is killed! (The same holds for `=.basis`.)

`+band...`, `+bweights...` are created if the band-structure/band-weights plotting options have been set via FEDIT. They contain the band-structure/band-weights. Use XFBP `filename` (or before version 14 FEDIT `-bandplot`⁵) to produce the related pictures from them. Since version 14, a suffix is appended to `+b...` files (Table 3.1). Also since version 14, two band weight files are produced in full-relativistic mode, one with $jl\mu$ -orbital projection and one with (approximate) $lm\sigma$ -projection (see Sec. 8). When unfolding is active, both default and unfolded files will be created.

cases	not full-relativistic	full-relativistic
default	<code>+band</code> <code>+bweights</code>	<code>+band</code> $lj\mu$ <code>+bweights</code> $lm\sigma$ <code>+bweightslms</code>
<code>=.kp</code> meshes (Fermi surface)	<code>+band_kp</code> <code>+bweights_kp</code>	<code>+band_kp</code> $lj\mu$ <code>+bweights_kp</code> $lm\sigma$ <code>+bweightslms_kp</code>
unfolding	<code>+band</code> <code>+bweights</code> <code>+bweights_unfold</code>	<code>+band</code> $lj\mu$ <code>+bweights</code> <code>+bweights_unfold</code> $lm\sigma$ <code>+bweightslms</code> <code>+bweightslms_unfold</code>
<code>=.kp</code> and unfolding	<code>+band_kp</code> <code>+bweights_kp</code> <code>+bweights_kp_unfold</code>	<code>+band_kp</code> $lj\mu$ <code>+bweights_kp</code> <code>+bweights_kp_unfold</code> $lm\sigma$ <code>+bweightslms_kp</code> <code>+bweightslms_kp_unfold</code>

Table 3.1: Band weights files.

If the XFPLO `=.bwdef` mechanism (Sec. 9.3) is used the resulting weights file is named by the user.

⁵This will use BANDPLOT to create a Postscript file.

It is always a good option to check which files got created after a run (`ls -ltr`).

+points Created in the initialization phase at the beginning of the FPLO run. It contains the special symmetry points used for the band structure creation. It is used by XFBP (old: BANDPLOT).

+symmetry Created by the symmetry module of FPLO. It contains information about the crystal symmetry.

+fcor.sort.spin These files are created, if option PLOT_BASIS is switched on. They contain the radial part of the core orbitals.

+fval.sort.spin These files are created, if option PLOT_BASIS is switched on. They contain the radial part of the valence orbitals. Here valence orbitals include all non-core orbitals.

+fkval.sort.spin and +fcor.sort.spin The kinetic core and valence functions. This is supplied mainly for debugging purposes.

+atpot.sort.ivat (New since version 7.00) Created if option PLOT_REALFUNC is switched on. Contains the effective potential, used in the Hamiltonian of the radial atom like equation for the calculation of the basis orbitals

$$V = \frac{l(l+1)}{2r^2} + \langle V \rangle_{\text{spherical}}$$

$\langle V \rangle_{\text{spherical}}$ is a spherical potential defined by the basis parameters. If two basis orbitals have the same parameters, they belong to the same class and have the same atomic potential. The index `ivat...` labels the different classes. These files are supplied for debugging purpose.

+dens.site.spin, +har.site.spin Created if option PLOT_REALFUNC is switched on. All these files contain the angular momentum components. They contain

- The density contributions of site 'site'.
The $L = 0$ part is multiplied with $\sqrt{4\pi r^2}$. So, it integrates to the electron number belonging to the respective site.
- The local multipole-neutral Hartree potential contributions of site 'site'.

Be aware, that the local contributions have no physical meaning, only the sum of all has one. These files are supplied for debugging purpose.

+fedit, +fedithelp Communication files between the executables FPLO/ BANDPLOT/ DIRAC and FEDIT. These files are created on every run of the executables. In the initialization sequence of the FEDIT run, they are deleted and the appropriate executable is called, to recreate them. In this way it is assured, that FEDIT always reads the correct information.

+plasmon This file contains the main axis and energies of the plasmon tensor.

+grid_dens.***

Obsolete since version 18. Instead see page 29. These files are created by the grid-output module if the open-dx option is set, which is now off by default. See FEDIT help in the grid-output sub-menu.

+voronoi (Obsolete since version 6.00) Created in the initialization phase at the beginning of the FPLO run. Contains the voronoi cell geometry.

If 'build voronoi' is true, +voronoi is created and used. If 'build voronoi' is false, one may specify 'voronoi file' as an alternative file to read from. This serves for cell merging. However, cell merging is not expected to be needed in versions later than 3.00.

+symanalysis (Discontinued since version 6.00) Created by the symmetry module of FPLO. It contains information about the crystal symmetry induced conditions for the (non relativistic) matrix elements of the onsite blocks of the density-matrix/Hamiltonian/overlap-matrix. The diagonal elements give the conditions for the site and angular momentum resolved DOS.

+fdval.sort.spin (Discontinued since version 6.00) These files are created, if option PLOT_BASIS is switched on. They contain the radial part of the derivative of the valence states with respect to the prefactor (λ) of the confining potential

$$V^{cf} = \lambda \left(\left(\frac{r_{NN}}{2} \right)^{\frac{3}{2}} \right)^{-4} r^4$$

$$\lambda = x_0^{-4}$$

+atcor.sort.spin, +atval.sort.spin (Discontinued since version 6.00) Created if option PLOT_REALFUNC is switched on. Contains the effective potential, used in the Hamiltonian of the radial atom like equation for the calculation of the basis orbitals

$$V = \frac{l(l+1)}{2r^2} + \left\langle \left\langle V^{\text{cryst.}} \right\rangle_{\text{spherical}} \right\rangle_{\text{smoothed}}$$

$\langle V^{\text{cryst.}} \rangle_{\text{spherical}}$ is the spherical average of the crystal potential around the considered site. The potential actually used (and contained in the files) is a smoothed version of V !

`+dirsh` (**Obsolete since version 6.00**) Created, if shape test is performed. It contains the shape function along the lines specified in the shape sub menu of FEDIT.

`+unity` (**Obsolete since version 6.00**) Created, if shape test is performed. It contains the sum of all shape functions minus 1 along the lines specified in the shape sub menu of FEDIT.

It should contain zeros, at least near the origin. (For larger distances the summation of all shape functions may be incomplete.)

`+densgrid` See `=.densgrid`

3.4 File class 3

These files have no prefix.

`grid_...` These are the output files for gridded data, which can be loaded into XFPLO like this:

```
xfplo =.in grid_dens.001
```

For more details read the XFPLO help screen in Sec. ??.

`wfdata...` These are the output files for the real space representations of the Wannier functions , which can be loaded into XFPLO like this:

```
xfplo =.in wfdata001
```

Or if you used the “use data directories” FEDIT option:

```
xfplo =.in +wfdata/wfdata001
```

For more details read the XFPLO help screen in Sec. ??.

`dmatedit.ini` DMATEDIT saves the local axes and stuff in this file.¹¹

`area_vs_angle.cmd`

This file is created by FDHVA and can be loaded with the help of XFBP. This produces a picture of the dHvA spectrum belonging to the iso surfaces treated in the latest FDHVA run.

`grid_***.[net|general|cfg]`

Obsolete since fplo18. Instead see page 29. If the `open-dx` option in the `fedit grid-output` submenu is set they are still created. These files can be created by the `grid-output` module. They define the gridded data structure and an `opendx` program to display it in `opendx`. This is explained in the `grid` sub-menu.

`bravais.ps, primitive.ps` Created in the initial part of the FPLO run, before the density is read. Contains a postscript picture of the bravais/primitive cell.

`vcell{site}.ps` (**Obsolete since version 6.00**) Created in the initial part of the FPLO run, before the density is read. Contains a postscript picture of the voronoi cell of site 'site'.

3.5 Directories

FEDIT creates/uses a subdirectory `+tmp` in the directory where it was called, to perform input file updates. This directory may be deleted after use of FEDIT (or at the end of the calculations) In newer verions FEDIT tries to delete it itself.⁶

The dHvA module (Sec. 13) uses the subdirectory `dHvAdata` to store auxillary information.

The new FEDIT option use data directories will put many of the output data into separate directories for a tidier organization.

⁶Sombody will argue that we should use the systems `tmp` directory. May be. But our way of doing it is independend of the system settings.



Chapter 4

Basis states

4.1 General definitions

4.1.1 Orbital structure

The FPLO basis consists of atom-like orbitals Φ_{ti} for each Wyckoff position t with atom-like quantum numbers (qns) i , which are products of radial and angular functions. Depending on the mode of the relativistic treatment we have at atom t orbitals of shape

$$\Phi_{ti} = \begin{cases} \varphi_{tnl} Y_{lm} \chi_s & \text{without spin orbit coupling (NREL/SREL)} \\ \begin{pmatrix} g_{tnlj} \chi_{\kappa_{lj}\mu} \\ i f_{tnlj} \chi_{-\kappa_{lj}\mu} \end{pmatrix} & \text{with spin orbit coupling (FREL)} \end{cases} \quad (4.1)$$

where n is the main qn and χ_s are spin- $\frac{1}{2}$ basis spinors: $\sigma_z \chi_s = \chi_s s$ ($s = \pm 1$), Y_{lm} are (real) spherical harmonics with angular momentum qns l and $m \in [-l, l]$ and φ_{tnl} are radial functions (one per nl -shell and atom). (In scalar relativistic modes there are actually up to four radial functions per φ_{nl} -shell, the details of which are yet to be published.) In FREL mode the orbitals are four-spinors with two large $g_{tnlj=l\pm\frac{1}{2}}$ and two small $f_{tnlj=l\pm\frac{1}{2}}$ radial functions per nl -shell and $\chi_{\kappa_{lj}\mu}$ are spherical spinors with spin-orbit quantum number $\kappa = (2j+1)(l-j)$, total angular momentum qns $j = l \pm \frac{1}{2}$ and $\mu = [-j, j]$, such that $\hat{\kappa} \chi_{\kappa\mu} = -\chi_{\kappa\mu} \kappa$, $\hat{j} \chi_{\kappa\mu} = \chi_{\kappa\mu} j$, $\hat{j}_z \chi_{\kappa\mu} = \chi_{\kappa\mu} \mu$ and in detail $\hat{\kappa} = 1 + \hat{\sigma} \hat{L}$, $\kappa_{lj=l+\frac{1}{2}} = -(l+1)$, $\kappa_{lj=l-\frac{1}{2}} = l$.

Note, that φ_{tnl} does not depend on the spin index s , so the spin polarization of the final wave functions must be obtained via additional orbitals with the same qns (polarization-orbitals). More generally, any polarization effect of an imagined effective radial function due to some perturbation must be obtained via polarization-orbitals. (PO)

From an atom point of view polarization-orbitals are orbitals with higher angular-momentum- and/or main-qns. We use the term PO more loosely than quantum chemists would. In a solid the distinction becomes mute, the more orbitals we add, since a $4p$ is an $l+1$ -PO to the $4s$ -orbital but a $5s$ -PO (an $n+1$ -PO to $4s$) is an $l-1$ -PO to the $4p$ -PO.

Now, group all orbitals of the same l - (lj -) channel together and we get as possible valence orbitals s -, p -, d - and f - orbitals and if polarization-orbitals are included it just determines how many orbitals of each l -channel there are (see Fig. 4.1). Compare this to an angular momentum expansion of plane waves. Of course, also $l > 3$ channels exist. However, the resulting orbitals are quite high in energy, have a quite large Hilbert subspace ($2l+1$ -dimensional) and rather small effect. So, we try to get away with low- l polarization-orbitals.

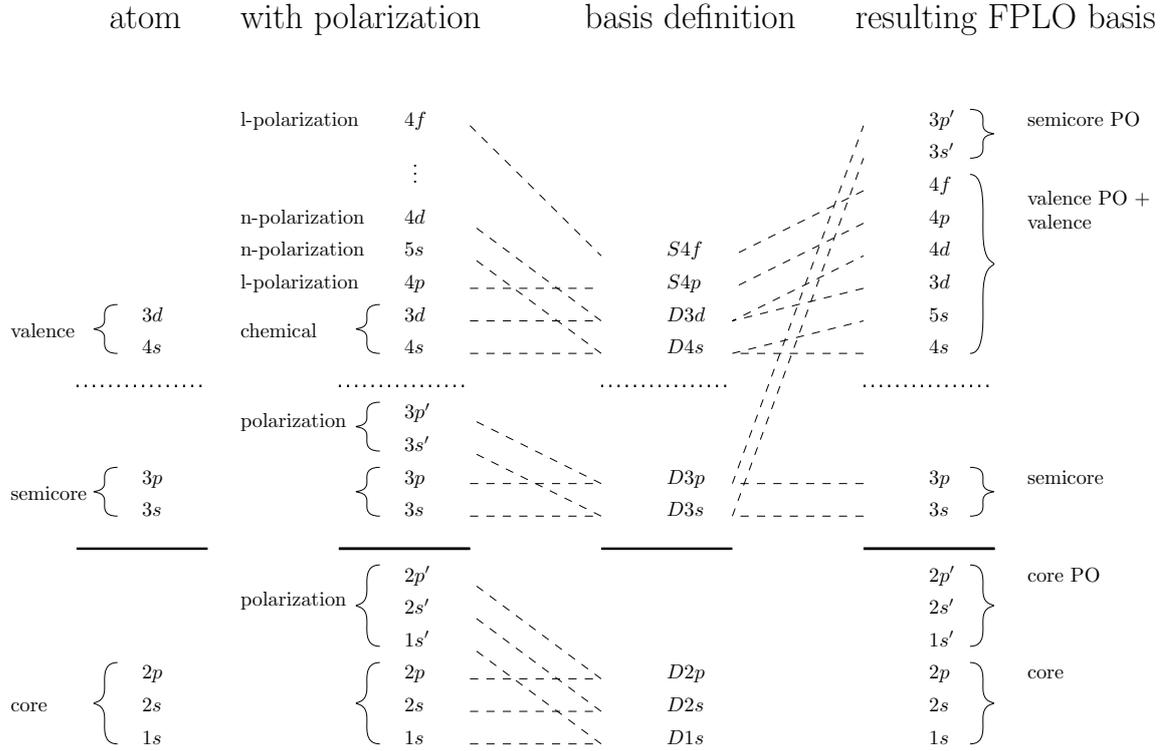


Figure 4.1: From atomic basis to FPLO basis: the left column shows the essential orbitals of an isolated atom. In the second column polarization orbitals are added. The third column is how one should think about the second column, while the right column shows the orbital order which results in FPLO.

In contrast to the valence orbitals, which serve to expand wave functions of higher and higher energy, the lower lying (fully occupied) semicore (core) states are described by a single atom-like orbital (such that the isolated atom limit) is correctly described. If polarization effects on these rather inert orbitals are important (hyper-fine field [B^{HF}] calculations) the corresponding polarization orbital needs to live in the same energy region as the main orbital and describe the effects of the polarization under consideration. To be specific if we need good spin polarization of the $1s$ -orbitals we could consider the two spin polarized atomic orbitals $\varphi_{n=1l=0s} Y_{00} \chi_s$, where $\varphi_{n=1l=0s}$ are the majority $s = 1$ and minority $s = -1$ $1s$ -orbitals of a spin polarized atom. Equivalent/similar sets of orbitals are a possibility, e.g. one spin-unpolarized orbital (and say the majority orbital). These two functions are quite similar, but after orthogonalization we get a proper $1s$ -basis of dimension 2 which spans a subspace of spin-polarized $1s$ -orbitals. What is important here is that we did not add a $2s$ PO (as we would do for the open valence section) but a $1s'$ -orbital with slight perturbation. After all, the $2s$ -orbital also exists in the basis. Since the core has a predetermined occupation pattern we need some bookkeeping to separate the first set of core orbitals from the polarization-orbitals by sorting the polarization-orbitals towards the end of the core-section and only occupy the first set. For the semicore section possible polarization-orbitals need to be sorted to the very end of the valence section for various reasons (see Fig. 4.1).

4.1.2 Orbital classification

The orbitals at each atom are classified into sections

core: orbitals which can be treated separately from all other orbitals since they

- are fully occupied
- have (essentially) *no* inter-site overlap/dispersion.
- have energies well below the chemical spectrum (valence band structure).

semicore: orbitals which need to be included in the non-core (valence) section since they

- have small inter-site overlap/dispersion
- although they are fully occupied
- and have energies below the chemical spectrum (valence band structure).

valence: orbitals which

- have large inter-site overlap/dispersion and form the valence and conduction bands
- are significantly partially occupied (chemical valence+*l*-polarization)
- are essentially un-occupied and form high-lying bands (polarization orbitals)

The differentiation into semi-core and valence is used in automatic Wannier basis determination and in the order of on-site orthogonalization: first the semi-cores are orthogonalized among themselves, then the valence orbitals are made orthogonal to the semicore section and then among themselves. Exception occur if multi-semicore orbitals are defined, in which case the first orbital of the multi-orbital is in the semi-core section and fully occupied, while the additional (nearly empty) orbitals are sorted towards the end of the valence section. Otherwise, the orthogonalization process would deform the valence orbitals of the same *nl*-qns. But we want to keep the shape of the valence orbitals as close to an atomic orbital with corresponding qns as possible. Also adding a semicore polarization orbital should not change the already existing valence orbitals.

4.1.3 Radial functions

The radial functions φ_{tnl} or g_{tlnj}/f_{tlnj} in Eq. (4.1) are solutions to a Schrödinger/Dirac equation with a modified atom potential $V_{tnl} = V_t^{\text{at}} + V_t^\infty + V_{tnl}^{\text{conf}} + V_{tQS}$, where *t* denotes the Wyckoff position (sort) and

- V_t^{at} is the self consistent atom potential, obtained by solving a DFT atom with confining potential $V_t^\infty + V_{tnl}^{\text{conf}}$.
- V_t^∞ is an infinite potential well outside the compact support radius ρ , which is individual for each atom and depends on the atom environment in a complicated manner. For FREL mode only the large component feels this well, which gives leaking small component at $r = \rho$, which however is a very tiny error.
- $V_{tnl}^{\text{conf}} = \left(\frac{r}{(P P_{nl} \rho)}\right)^N$ is a confining potential with atom-wide compression parameter *P*, orbital-specific compression parameter P_{nl} , which is made to be close to 1 (default value), and with compression power *N* (default: $N = 14$).
- V_{tQS} is added to the self consistent potential, when calculating the radial functions, and is obtained after self consistency by recreating the atom potential (for each orbital separately) with modified atomic occupation numbers such that the resulting atom is missing charge of amount Q_{nl} and/or has total magnetic moment S_{nl} . If *Q* is larger than the total atom charge (light atoms) a suitable exponentially

localized potential is used. In FREL mode both j -sub-shells use the same Q -, S -parameters. The S_{nl} option is experimental and was used in the context of the unofficial hyper-fine field (B^{HF}) module, where core and semi-core s -orbitals need to describe the spin polarization at the nucleus extremely well. If $S > 0$ the majority orbital for the corresponding qns is used, and if $S < 0$ the minority orbital (which has a different shape) is used.

4.1.4 Multi orbitals

To increase variational flexibility, polarization orbitals can be added, which results in the concept of multi-orbitals (MuO) (see Fig. 4.1). These are denoted by prepending a multiplicity character to the orbital name. Currently, multiplicities up to six are implemented with character: *SDTQUX* which stand for *Single*, *Double*, *Triple*, *Quadruple*, *qUintuple*, *siXtuple*.

A multi orbital $T3d$ then means that besides the first $3d$ -orbital two more d -orbitals are added to the basis, which of course will have their own Q -, P - and S -parameters. Which qns the orbitals carry depends on the orbital classification (see below and Fig. 4.1).

All orbitals of a multi-**core** and multi-**semicore** orbital will have the **same** qns as the first orbital, since we want to add slightly distorted orbitals with the same qns to add flexibility in the resulting effective radial functions (implicit result of diagonalization). So a double core ore semicore orbital expands as

$$D2p \rightarrow 2p_1 \ 2p_2$$

The orbitals of a multi-**valence** orbital will have **increasing main** qns, such that the resulting set of orbitals looks like a set of atomic orbitals for ever higher energies:

$$T3d \rightarrow 3d \ 4d \ 5d$$

4.1.5 Default basis

Using our classification we can define a save core section in the sense that the inter-site overlap is negligible even for close packed solids under pressure, say up to 200-300GPa.

The valence orbitals contain the “chemical” valence orbitals and one or two polarization orbitals (POs) of the lowest unoccupied shells (in an atom) with different angular momentum. Chemical valence is roughly defined as the HOMO+LUMO or HOMO-1+HOMO of the atom, (exceptions apply). Additionally the “chemical” orbitals are doubled to become double-orbitals (MuO with multiplicity 2), thus adding variational flexibility to the corresponding nl -channel.

Examples:

- H has chemical valence $1s$, to which we add the lowest lying $l \neq 0$ PO $2p$. Then we double the chemical valence by adding $2s$ resulting in $D1s$, $S2p$.
- C has chemical valence $2s$, $2p$ (HOMO-1+HOMO), to which we add the lowest lying $l \neq 0, 1$ PO $3d$. After doubling the chemical valence we get $D2s$, $D2p$, $S3d$.
- Na has chemical valence $3s$, $3p$ (HOMO+LUMO) to which we add the lowest lying $l \neq 0, 1$ PO $3d$. Then we double $3s$ and $3p$ to obtain the valence section $D3s$, $D3p$, $S3d$.
- Fe has chemical valence $4s$, $3d$ and the lowest lying PO with different l is $4p$. We double $4s$ and $3d$ to obtain the valence section $D4s$, $D3d$, $S4p$.

- Eu has chemical valence $6s, 5d, 4f$ and lowest lying polarization orbital with different l is $6p$. We double $6s, 5d$ and $4f$ to obtain the valence section $D6s, D5d, S6p, D4f$.
- La has chemical valence $6s, 5d, 4f$ and lowest lying polarization orbital with different l is $6p$. We double $6s, 5d$ to obtain the valence section $D6s, D5d, S6p, S4f$.
La is a bit of an exception since $4f$ is not doubled. The reason is that $4f$ should be essentially empty (which it is not in LDA). This might be less optimal and be changed in the future. It can also be changed via modifications.

According to these definitions the semicore section is formed by the remaining orbitals between the lowest valence and the highest core orbital.

In the default basis all core and semicore orbitals are single.

4.1.6 Parameters

The parameters which determine the basis are the atom specific compact support radius ρ , the atom-specific compression parameter P and compression power N . ρ and P are determined internally (based on optimizations performed once and for all) while N is fixed to its default $N = 14$.

The orbital specific charges Q_{tnl} (and/or spin moments S_{tnl} , currently experimental) and compression modifiers P_{nl} (see Par. 4.1.3) were also determined by optimizations and are tabulated internally. However, they can be changed if circumstances make this seem reasonable. They also have to be chosen, when the basis is modified (extended).

The default parameter values are $Q_{nl} = S_{nl} = 0$ and $P_{nl} = 1$. S is not currently used; it is an option for hyper-fine (B^{HF}) field calculations (experimental), in which case the spin polarization of s -orbitals at the nucleus must be very precise. Making core and semicore MuOs, with the added orbitals having $S \neq 0$, will improve the resulting spin polarization at the nucleus.

It seems to be the right strategy to let the essential orbitals have $Q = S = 0$ in order to get atomic (yet compressed) orbitals with energies such that the single-atom limit is basically correct. Essential means: first core/semicore and first chemical valence orbitals, basically the orbitals which are occupied in the atom. Imagine calculating a free standing atom, then compression goes to zero (ρ becomes large) and these orbitals need to be solutions to the free atom, hence $Q = S = 0$.

For orientation:

- the first core orbitals (usually core is single) have default values
- the first semicore orbitals (usually semicore is single) have default Q and S but may have $P \neq 1$ (from optimization in solids), especially for heavier atoms under pressure.
- The chemical valence orbitals (first of MuOs) usually have default $Q = S = 0$ but some $P \neq 1$.
- The additional orbitals of a MuO and the true polarization orbitals usually need $Q \neq 0$ (determined by optimization and tabulated).
- *Experimental*: additional orbitals for core and semicore MuOs (for B^{HF}) should use $Q_{2\dots} = 0, 2, 4, \dots$ and $S_{2\dots} = 5, -5, 10, -10$ (remember $Q_0 = S_0 = 0$), where $Q_i = 0$ seems to be good for lighter atoms and $Q = 2$ for heavier. If problems occur, they are visible in the population analysis, where the net occupations become significantly larger than the gross occupations. Example: $D2s$ $Q = (0, 0)$, $S = (0, 5)$ or $D2s$ $Q = (0, 2)$, $S = (0, 5)$. This business is prone to instabilities.

4.1.7 Modifications

The default basis can be modified in various ways: using FEDIT, using `pyfplo.fploio.Basis` (`./pyfplo/pyfplo.pdf`) or editing `=.basdef` by hand (see Sec. 4.4). The following are the most important modifications

- Extend the basis
 - add more polarization orbitals to existing orbitals (increase the multiplicity of multi-orbitals)
 - add a d -orbital if none exists (H and He)
 - add an f -orbital if none exist
- Core treatment of $4f$ (for crazy people who know what they do)
 - move $4f$ into core
 - move $4f$ into core and remove all remaining f -POs from the valence altogether.
- Experimental: Improve core/semicore for B^{HF}
 - double/triple the core
 - double/triple the semicore
- Modeling, testing ... (not accurate)
 - make all valence orbitals single (not a standard option, but possible using PYFPLO)
 - remove orbitals (possible using PYFPLO)

4.2 Basis setup

As explained in Sec. 4.1, the default basis is out of the box and ready to go. However, it turns out that in some contexts a modified (especially extended) basis gives better results. To understand how such modifications are achieved we need to explain the basis setup of FPLO in detail.

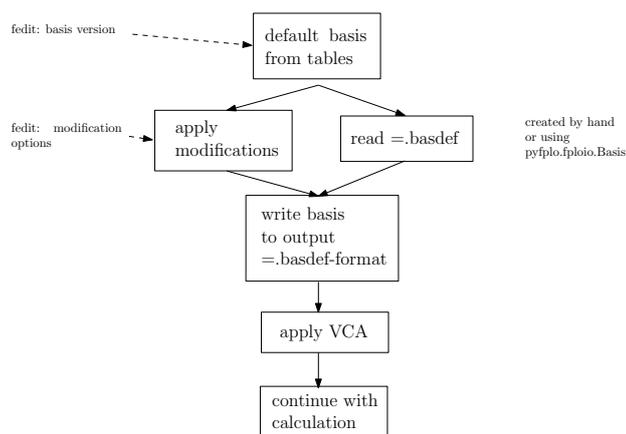


Figure 4.2: Flow of basis setup on FPLO run.

When FPLO is running

1. for each Wyckoff position the default basis is extracted from internal tables according to a user selected basis-version (currently there is still only one version available).
2.
 - (a) If standard modifications are requested by the user these are applied.
 - (b) If the file `=.basdef` is found in the working directory it will be read and it's content will overwrite the default basis and sidestep the FEDIT-defined modifications!
3. The basis obtained thus far is written in `=.basdef` format to the output in a section marked like this:

```

-----
Start: content of =.basdef
-----
...
...
...
-----
End  : content of =.basdef
-----

```

4. If the virtual crystal option was chosen (VCA) the basis obtained thus far will be merged with a default basis of appropriate atomic number to interpolated the basis parameters (as good as possible) between the two bases belonging to the interval boundaries of the atomic number interval which contains the VCA-nuclear charge. This process cannot be made absolutely consistent, since neighboring atoms can have differing basis sets, e.g. Zn and Ga. So keep in mind that VCA interpolation is applied after the basis definition/reading of `=.basdef`! This also means that VCA will not be completely symmetric in the sense that Zn_xGa_{1-x} with Zn as nominal (input) element will be slightly different from Zn_xGa_{1-x} with Ga as nominal (input) element. Plan your investigations accordingly. VCA is a crude approximation after all.

Note, that there are two mutually exclusive routes to modifying the basis

- use-specified modification via FEDIT
- whatever is written in `=.basdef`

Also note, that starting from FPLO version >21.00, the `=.basdef` content in the FPLO output is no longer the preferred starting point to created a user-defined basis. Instead use `pyfplo.fploio.Basis` (and various examples) explained in [../pyfplo/pyfplo.pdf](#).

Some users used `=.basdef` in the past as the only way to modify the basis. However, it is advised to use the newer mechanisms for basis manipulation in future projects. In almost all cases the modifications provided by `FEDIT/pyfplo.fedit.Fedit` should be enough.

4.3 `=.basdef`

Although `=.basdef` is not the preferred path to modifying the basis it will be explained first, since it is a useful representation of the basis to discuss extensions in detail. (It actually is the internal representation.)

The file `=.basdef` contains one line for each Wyckoff position (sort), which looks like:

```
3 P=(0.67); N=(14); 1s2s2p : 3s S3p P=(0.79); / D4s Q=(0,3.1) P=(0.92,1.01) ; D3d ... S4p Q=(4.2) P=(1);
```

and has the structure:

```
sort P=(); N=(); core : semicore / valence
```

- The first number is the sort number. Hence, if `=.basdef` does not contain certain sorts, the basis definitions of these missing sorts will retain their default settings.
- The leading `P=()` defines the atom-wide compression parameter P_t (see Secs. 4.1.3,4.1.6).
- The leading `N=()` defines the compression power N_t .
- Between the power and `:` follows the core section
- Between `:` and `'/'` follows the semicore section
- After `'/'` follows the valence section
- An orbital with default parameters $Q_{nl} = S_{nl} = 0$ (and $P_{nl} = 1$ for non-core orbitals) can simply be written with the usual notation `2p`.
- A single orbital with non-default parameters must be written `S3p P=()`, where only the non-default parameters need to be explicitly specified. Of course all parameters can be specified, which for this example would read `S3p Q=() S=() P=()`;
- A multi-orbital starts with a multi-orbital character (Sec. 4.1.4) followed by the orbital name and the parameter list, which can consist of Q , S and P (if not core). The number of values in the parameter list must match the multiplicity of the multi-orbital, e.g. `T3d` requires `Q=(Q1,Q2,Q3)`.
- For the core a condensed notation like `D{1s2s2p} Q=(Q1,Q1) S=(S1,S2)`; can be used to group several core orbitals together, having the same parameters.

4.4 Basis modifications

4.4.1 In FEDIT

Some standard basis modifications are implemented in the FEDIT menu `Basis` since FPLO version 22. The modifications are applied to the default basis in the order as they appear on the menu screen! This together with the explanations below determines the resulting basis. Naturally, all calculations of one (sub-) project should use the same basis! So, using simple python scripts (`pyfplo.fedit.Fedit`) to create all inputs seems advisable.

Modifications:

basis-version identifies a complete set of default basis definitions for all (implemented) atoms. As of version 22, only one basis version is implemented, but others might follow.

extension-level determines how many valence polarization orbitals will be added to the default basis. The default level is 1 and yields the default basis with certain predefined valence MuOs. Using the `=.basdef` notation (Sec. 4.3) a valence section example could look like this:

```
... / D4s Q=(Q4s,Q5s) P=(P4s,P5s); D3d Q=(Q3d,Q4d) P=(P3d,P4d); S4p Q=(Q4p) P=(P4p)
```

which expands to the valence orbitals

```
4s5s3d4d4p
```

We used place holders for the parameter values. Note, that we name the values Q4s, Q5s, ... according to the rules of Sec. 4.1.4.

An extension level 2 would increase the multiplicity of all MuOs by one (S→D, D→T, ...), level 3 by two and so on, such that our example for extension level 2 would yield

```
... / T4s Q=(Q4s,Q5s,Q6s) P=(P4s,P5s,P6s); T3d Q=(Q3d,Q4d,Q5d) P=(P3d,P4d,P5d); D4p Q=(Q4p,Q5p)\
P=(P4p,P5p)
```

which expands to the valence orbitals

```
4s5s6s3d4d5d4p5p
```

The increased multiplicity requires that all parameter lists get a new value appended. Since only the default basis parameters were optimized and since the optimization becomes less stable the more parameters are added, we need a rule to chose additional parameters, which is

$$\begin{aligned} Q_{n+1,l} &= Q_{n,l} + 2 \\ P_{n+1,l} &= \max(0.85, \min(P_{nl}, 1)) \end{aligned} \quad (4.2)$$

add-3d The default basis for H and He does not contain a *d*-orbital. This options adds a single S3d Q=(5) P=(1); orbital to the basis, if the atom does not contain any *d*-orbital in any section.

add-f If the valence section of an atom does not contain an *f*-orbital this option adds a single Snf Q=(5) P=(1); , where n is the lowest *f*-main qn, not contained in any section. So, if there is no *f*-orbital at all this adds a S4f and if the core contains a 4*f*-orbital this adds a S5f. If the valence contains an *f*-orbital none is added.

Put-4f-into-core uses a list of elements and/or a list of sorts as input. For some reasons it might be desirable to put the 4*f*-valence orbital into the core and give it a particular (non-full) occupation (FEDIT submenu Core-Occupation). The union of the sorts resulting from the two input lists defines for which atoms this happens. This option moves an existing 4*f*-orbital to the core and leaves the remaining *f*-POs in the valence. Such a calculation, even if it does not have a bad population analysis (net- much larger than gross-occupation), will have unreliable total energies! So, only use this option if you need to get rid of the *f*-states from the Fermi energy and if you know what you are doing. I would use LDA+U instead! Example: Eu has default basis

```
core : semicore / non-f-valence D4f Q=(Q4f,Q5f) P=(P4f,P5f);
```

which applying this option becomes

```
core 4f : semicore / non-f-valence S5f Q=(Q5f) P=(P5f);
```

Put-4f-into-core-and-remove-f-POs uses a list of elements and/or a list of sorts as input. This option puts any existing 4*f*-valence orbital into the core and removes the remaining 4*f*-POs of the corresponding MuO altogether. This option might be more stable then the previous one, since a (physical valence) 4*f*-orbital in the core violates the zero inter-site overlap condition and the additional *f*-valence POs of the corresponding *f*-MuO will lead to wrong matrix elements between the core-4*f* and the remaining *f*-valence POs. By removing the POs this error is removed (other similar errors remain though). The closing remarks of the previous option apply. Eu has default basis

```
core : semicore / non-f-valence D4f Q=(Q4f,Q5f) P=(P4f,P5f);
```

which applying this option becomes

```
core 4f : semicore / non-f-valence
```

Multi-core-orbitals uses a multiplicity and a Q - and S -list of said multiplicity as input. The default are empty lists, which indicate “no change”. This option will turn all core orbitals into corresponding multi-core orbitals each having the specified Q - and S -parameter values (experimental for B^{HF} calculations). Example: multiplicity 2 with two Q - and S -values turns a core section

```
1s2s2p / ...
```

into

```
D{1s2s2p} Q=(Q1,Q2) S=(S1,S2); /...
```

Multi-semicore-orbitals uses a multiplicity and a Q - and S -list of said multiplicity as input. The default are empty lists, which indicate “no change”. This option will turn all semicore orbitals into corresponding multi-core orbitals each having the specified Q - and S -parameter values and P -values copied from the P -value of the existing single orbital (experimental for B^{HF} calculations). Example: multiplicity 2 with two Q - and S -values turns a semicore section

```
core : 3s S3p P=(0.79); / valence
```

into

```
core : D3s Q=(Q1,Q2) S=(S1,S2); D3p Q=(Q1,Q2) S=(S1,S2) P=(0.79,0.79); / valence
```

which in turn gets internally expanded according to the rules of Sec. 4.1.1.

If you want to check which basis actually resulted from the modifications have a look at the `=.basdef` section in the FPLO output which was explained in Sec. 4.2.

In practical applications we have used extension level 2, add-3d and add-f together as a kind of “standard extended basis”, if such a basis seemed to be indicated by enhanced accuracy requirements or for testing that the default accuracy was indeed sufficient.

In publications one should report the use of basis modifications. Assuming a “standard extended basis” was used we suggest a formulation like:

```
... we used the default basis with extension level 2 and compulsory valence  $f$  and  $d$ -orbitals
...
```

or if more than one basis versions exist in the future

```
... we used the FPLO9 basis with extension level 2 and compulsory valence  $f$  and  $d$ -orbitals
...
```

where FPLO9 stands for the basis version descriptor as found in the FEDIT basis menu.

When in doubt list the orbitals explicitly.

4.4.2 PYFPLO

As an advanced option the `=.basdef`-method of modifying the basis, using class `pyfplo.fploio.Basis`, is outlined in the following.

You will need the basis-version identifier (as for FEDIT) on which to base the modification and a list of the elements or atomic numbers of all Wyckoff positions (explained in [../pyfplo/pyfplo.pdf](#)). With this you can extract the default basis directly from PYFPLO (without running FPLO first, as in the past). Then you can modify the basis for each sort at will (as provided by the python interface) and save it into `=.basdef`. This way one can setup the FPLO input (`=.in`) together with the basis (`=.basdef`).

Let us stress again that the preferred way to achieve a non-default basis is to use FEDIT or in scripts `pyfplo.fploio.Fedit` (if it provides what the user needs)!

Of course one can also create a default `=.basdef` via PYFPLO (or extract the basis from an output file via `fout2in -b`) and then edit it by hand as was done in the past.



Chapter 5

Algorithms

5.1 SCF k -mesh

Traditionally, the k -integration grid is obtained by subdividing the primitive cell into micro cells (primitive cell algorithm: PCA). The resulting grid-points are reduced with respect to symmetry to reduce the number of k -points as much as possible. This has the side effect that in centered lattices one cannot chose different subdivisions in directions of different length. (Anyhow, the primitive vectors have equal length in centering directions.)

To remedy this, a new scheme, the conventional cell algorithm (CCA), has been implemented. It takes the conventional reciprocal cell and subdivides it in all three directions individually (if symmetry allows it). So, for instance in body centered tetragonal lattices $N_c \neq N_a$ but $N_a = N_b$ can be chosen. In body centered orthorhombic lattices all subdivisions can be different! In order to be able to get a number of points in the BZ as small as possible (reduction by primitive translations) the subdivisions are internally multiplied by integer factors depending on the centering: 2 for body- and face-centered and for the centering plane of base-centered and 3 for all directions of rhombohedral lattices. This also means that a subdivision of 1 will result in a minimum of 2 or 3 subdivisions of the conventional cell, which after back-folding into the primitive cell (BZ) gives a smaller number of points, of course. These multipliers lead to the effect that an input of say 12 12 12 results a in larger number of points in the BZ than in PCA. For non-centered lattices both algorithms are identical.

The algorithm is chosen in the FEDIT main menu under hotkey M (BZ subdiv (M)ethod).

In this context an automatic subdivision was also implemented in version 19.00. The input `k-mesh subdivision` in the FEDIT main menu can have the following meaning now:

- three numbers > 1 , e.g. 12 7 5 define the individual subdivisions of the primitive cell (PCA) or of the conventional cell BEFORE centering multipliers are applied (CCA).
- A subdivision of 1 at any position fixes this subdivision to 1 (effectively 1, 2 or 3 in CCA, depending on the centering), even if auto subdivision is applied to other directions. This is ment to signal to the program that there is no dispersion in this direction (momentarily this is used by some parts of the Wannier function module).
- A subdivision of 0 indicates auto subdivision:
 - All subdivisions of all directions, which are not > 1 , will be multiplied to obtain the total number of points N_{BZ} in the BZ.

- The directions which are not fixed (subdivision \neq 1) will be subdivided such that maximal isotropy of the resulting grid in these directions is achieved and that the resulting number of points in the BZ is approximately N_{BZ} . So, if some directions are fixed the grid is not isotropic in these fixed directions compared to the others.
- Isotropy is followed over achieving N_{BZ} , which means there could be other resulting subdivisions which get closer to N_{BZ} but are less isotropic. Consequently, the actual number of points N_{BZ} must be inquired from the output file: search for “irreducible k-points” to get something like

```
BZMESH: 84 irreducible k-points from 2197 ( 13 13 13 )
```

for the PCA and

```
BZMESH: 85 irreducible k-points from 2048 ( 16 16 16 ) / 2 from input ( 8 8 8 )
```

for the CCA. The meaning is this:

- * there are 85 irreducible points (reduced by symmetry and primitive translations)
- * and 2048 points in the smallest primitive cell (or BZ) obtained from the input subdivisions 8 8 8,
- * which because of centering (fcc) get multiplied by 2 in each direction (hence the subdivisions 16 16 16 of the conventional cell)
- * which leads to $\frac{1}{2}16^3 = 2048$ points in the BZ, since the primitive reciprocal lattice is bcc which has half as many points as the conventional reciprocal cell (\rightarrow divide by 2).
- Examples:
 - * 1 1 1: triggers Γ -point mode (temperature broadening, as in molecule mode)
 - * 0 1 0 or 0 0 0 or any combination of ones and at least one zero: the actual 1 1 1 subdivision using the tetrahedron or Methfessel-Paxton method.
 - * 2000 0 0: auto subdivide all directions to get approximately 2000 points
 - * 200 10 0: same ($N_{\text{BZ}} = \prod_{i|N_i \neq 0} N_i$)
 - * 1 500 0 or 1 0 500: auto subdivide the *b*- and *c*-direction to get a total of 500, which in CCA can mean 2 or 3 subdivisions in the conventional *a*-direction (see above).
 - * 12 6 1 or 1 400 1: subdivide as is and indicate lack of dispersion in the directions with subdivision=1.

5.2 LSDA+U

The new basis since version 7 sometimes requires the gross population projector, since the new basis functions are more extended than the FPLO<5 basis functions. Due to historical reasons the default is still orthogonal projection (net population). Please consider the gross projection, especially if your calculations do not converge badly, especially for small volumes and cases of considerable ligand hybridization. Check the population analysis for overly large net occupations (much larger than gross). This might be a hint to use the gross-projector.

Now, full relativistic LSDA+U is implemented. Note, that this is less stable than non-full-relativistic LSDA+U. Main reasons are

- bands now carry non-integer spin weight, which allows reshuffling between spin occupations depending on tiny band shifts at the Fermi level
- symmetry is lower \rightarrow more non-zero occupation number matrix elements

- at the same time now orbital momentum is a meaningful shell property, which increases the number of solutions, between which to fluctuate. Furthermore, it seems to converge slower than the spin moment and total occupation.

In order to see the proper orbital momenta we have to project onto the spin quantization axis, since otherwise we can not compare S_z which is in the field coordinate system and L_z , which would be in the global coordinate system. This means that when using DMATEDIT you have to go into the proper system (axis dialog) to manipulate the orbital moment.

Note, that in non-spin polarized full relativistic LSDA+U time reversal symmetry is enforced.

5.3 Total energy

The total energy in the output (see Sec. 7 and an actual output file: search for EE) is the usual density functional zero temperature energy. In Broadening schemes the resulting energy is not force consistent (Weinert Davenport <https://doi.org/10.1103/PhysRevB.45.13709> and <https://doi.org/10.1103/PhysRevB.49.13975>) and an entropy term can be defined and added, which gives the electronic free energy $F = E^{\text{tot}} - TS$, which is force consistent. Alternatively, an approximate correction can be added to the total energy to obtain $\frac{F+E}{2} \approx E^{\text{tot}}(T=0)$ (see <https://doi.org/10.1103/PhysRevB.58.13459> and therein). This does not give corrected forces (whose correction are not implemented).

for historical reasons, in a molecule and Γ -point calculation the output under search string EE is the free energy while for all other BZ integration schemes it is the total energy, while the free energy and the corrected one are printed separately. All can be grepped using GREPPFLO (Sec. 7)

5.4 Hamilton matrix elements (three-center integrals)

In order to calculate the Hamiltonian matrix elements integrals $I = \langle \Phi_{\mathbf{R}'\mathbf{s}'} | V | \Phi_{\mathbf{R}\mathbf{s}} \rangle$ of two orbitals with the crystal potential need to be evaluated. In the old times these contained terms which were named three-center-integrals (THCI). This differentiation is no longer appropriate, since the technical solution to the integration problem does not separate into n -center terms, however for historical reasons the name is still used. The way these integrals are implemented requires shape functions (from a partition of unity) at each site $\mathbf{R} + \mathbf{s}$ in the lattice, which decompose each single integral into contributions from all sites in the lattice. This results in a large number of new integrals, centered at sites and restricted to the compact support of the shape functions around these sites. These are essentially single-site integrals. To evaluate them a numerical mesh is introduced which consists of a radial and an angular mesh. The angular mesh is chosen from a set of predefined Lebedev-type meshes of various sizes (not exactly Lebedev, since we determined them ourselves, including a set of hexagonal meshes, which are slightly less effective but have the correct symmetry!). The size of the mesh determines how many spherical harmonics are integrated exactly, or more practically, the larger the mesh size the more accurate the angular integral. Now, it seems reasonable to chose smaller meshes for radial mesh points with smaller radius. This is implemented such that depending on a fixed radius criterion the angular meshes for each radial point are picked from the set of meshes having sizes between $\text{min}_{\text{angmesh}}$ and $\text{max}_{\text{angmesh}}$, smaller sizes for smaller radii and larger sizes for larger radii. This means that there are jumps in the integration accuracy at certain radial mesh points. As long as the compact support radii do not change, i.e. as long as the lattice parameters are not varied, these jumps merely enter the all-over integration accuracy. But, if the radii change, jumps in angular mesh size can from one radial mesh point to the next, which leads to different numerical integration errors depending on the lattice parameters, which in turn produces discontinuities in the resulting energy curves with respect to lattice parameters/volume.

Now, in all our tests this turned out to produced numerical noise far smaller than the physically important energy differences. Only, in 2021 did we learn that there might be cases where this matters. For this reason the choice of the THCI-integration mesh was made adjustable in the `FEDIT Numerics` submenu (hotkeys N1...N4).

5.5 XC-functionals

FPLO implements a limited number of exchange and correlation (xc) functionals. Some need explaining, The xc-functional/potential is selected in the `FEDIT` main menu under hotkey V (`(V)xc-version`). For some xc-potentials certain parameters can be set in in `FEDIT` submenu `(X)C options`.

5.5.1 Becke-Johnson (BJ06) and modified Becke-Johnson (mBJ)

The BJ06/mBJ potentials are implemented in several modifications mentioned in the literature:

BJ06 original Becke Johnson 06 [2, 11]

BJ06LDAc original Becke Johnson 06 + Perdew-Wang-92 correlation [2, 11, 8]

mBJ modified Becke Johnson (parameter $c = -0.012 + 1.023\sqrt{g}$) [10]

mBJLDAc modified Becke Johnson + Perdew-Wang-92 correlation (parameter $c = -0.012 + 1.023\sqrt{g}$) [10, 8]

mBJlinLDAc modified Becke Johnson + Perdew-Wang-92 correlation version "All" (P-present) (parameter $c = 0.488 + 0.500g$) [4, 8]

mBJsemcoLDAc modified Becke Johnson + Perdew-Wang-92 correlation version "Semi Conductor" (P-semiconductor) (parameter $c = 0.267 + 0.656g$) [4, 8]

where

$$g = \frac{1}{V_{\text{cell}}} \int_{V_{\text{cell}}} \frac{1}{2} \sum_{\sigma} \frac{|\nabla \rho_{\sigma}|}{\rho_{\sigma}} d^3r$$

$$c_{\text{mBJ}} = A + Bg^e$$

For the users convenience c_{mBJ} can be set fixed in the xc-submenu of `FEDIT`. Both g and c_{mBJ} can be grep-ed from the output with `GREPPFLO` (Sec. 7)

Note, that all those xc-potentials are not derived from a functional and hence do not provide a total energy.

For stability (and conceptual) reasons we used slight technical variations as compared to the officially documented WIEN implementation, which might lead to small differences in the results. Another source of differences is the basis set. It turns out that the gap is not always best described by the default FPLO basis. In such cases an extended basis (Sec. 4) was used to obtain better results. Following the published results, we performed comparisons between WIEN and FPLO results for various compounds, which are shown in Figs. 5.1,5.2,5.3. (Note, that the occasional compound might be missing in either the FPLO or WIEN data). We get an all-over good agreement with WIEN when comparing the scatter with respect to experiment. However the scatter plots for FPLO-exp and WIEN-exp differ for certain compounds. These deviations lay on average within the accuracy of the xc-potential as indicated by the WIEN-exp scatter plot.

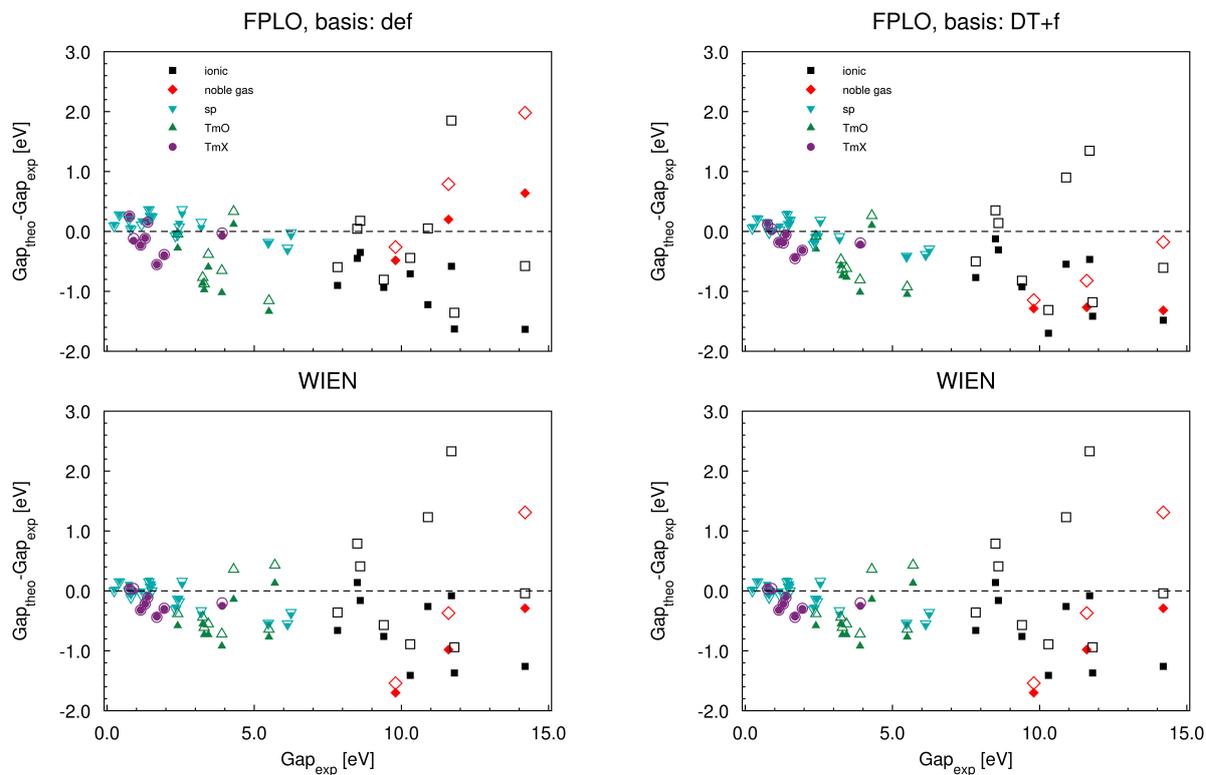


Figure 5.1: Gap error versus experimental gap: upper left: FPLO default basis, upper right: FPLO extension level 2 and compulsory f - and d -orbitals, lower panels: WIEN [4].

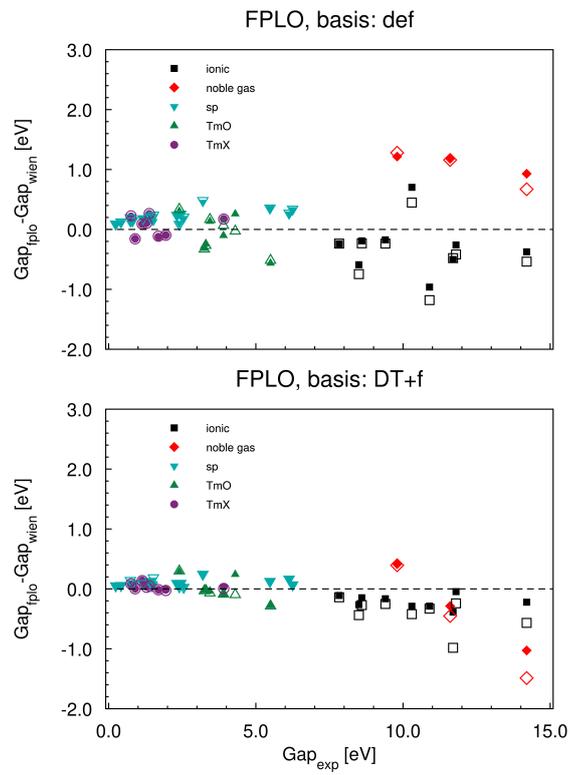


Figure 5.2: FPLO-WIEN gap difference as a function of the experimental gap: upper: default basis, lower: basis with extension level 2 and compulsory f - and d -orbitals. WIEN from [4].

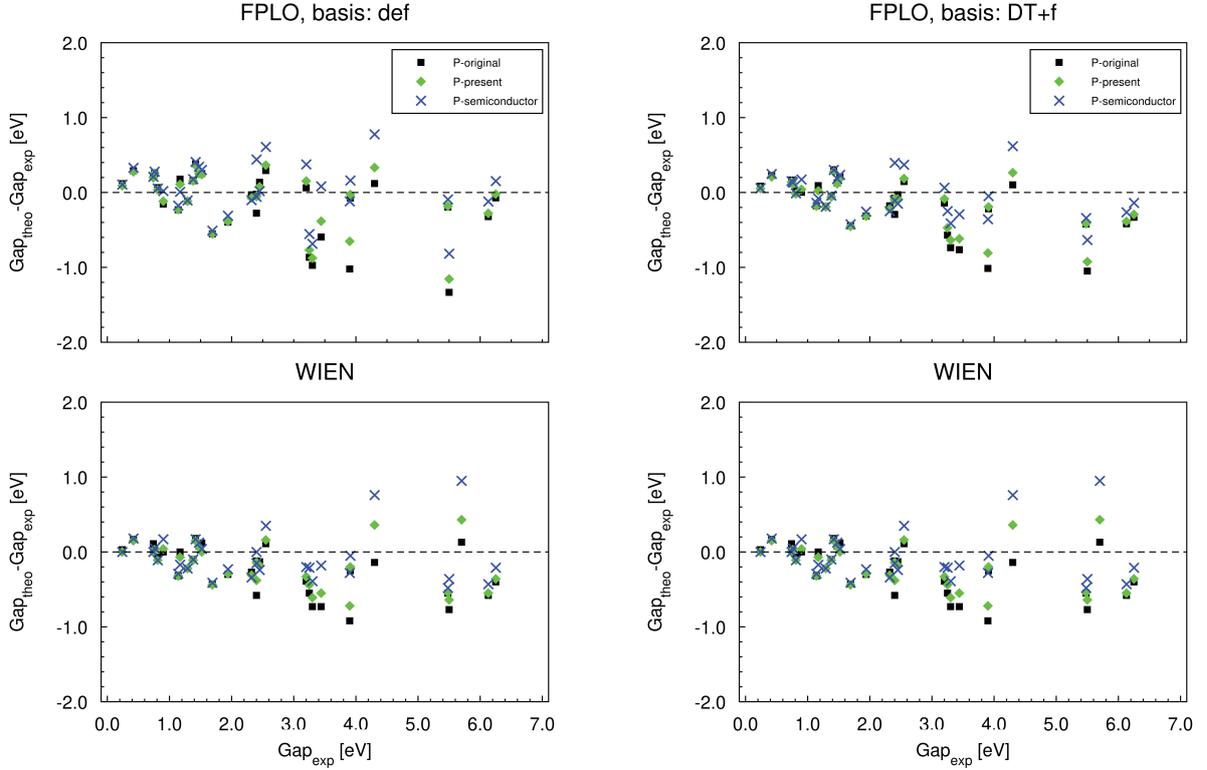


Figure 5.3: Comparison of the Gap error versus experimental gap for three mBJ flavors for smaller gap compounds : upper left: FPLO default basis, upper right: FPLO extension level 2 and compulsory f - and d -orbitals, lower panels: WIEN [4].

An example of mBJ calculations can be found in [../pyfplo/pyfplo.pdf](#) (mBJ XC-potential).

A notable difference between WIEN and FPLO is the definition of the kinetic energy density τ_σ . In WIEN it is derived from the band energies and the effective potential, while in FPLO it can be calculated directly from the application of the kinetic operator on the basis functions. The latter has the advantage that it straight forwardly generalizes to scalar-/full-relativistic modes and that it fits better into the general program flow of FPLO. In particular it improves iteration stability.

It has to be mentioned that BJ06 is a local potential, which tries to mimic the effect of a non-local xc-potential. This leads to a strong sensitivity of the xc-potential on the actual band occupation. Slightest re-occupations between the valence and conduction bands lead to remarkably large changes in the potential in the interstitial region, which in turn acts back on these occupations. This gives an inherently unstable situation, which shows in the bad SCF-iteration behaviour of this xc-potential. Additionally, for an isolated atom (or molecule) the asymptote of this xc-potential is formally $c_0 + \frac{c_1}{r}$ but is in practice determined by the ratio τ and ρ which both become very small at some radius, after which they are no longer numerically reliable. The resulting large errors in the asymptotics then reflect back on the self consistency. For this reason we established a smooth cross-over between the numerical potential for smaller radii and an analytic form of the asymptotic tail for the atom calculations (which underlie every FPLO calculation for the determination of the basis.)

To make matters worse the parameter g in c_{mBJ} in mBJ relies on an integral of a function containing the square root of $\left| \frac{\nabla \rho}{\rho} \right|$, which has nodal lines in the interstitial (and becomes unstable for large distances from atoms) which leads to kinks and a function which is difficult to integrate numerically. This conceptual flaw has

been remedied in FPLO by using a weighted integral, which screens away the regions with small ρ and τ . For the atom calculations a correct asymptotic of $\left|\frac{\nabla\rho}{\rho}\right|$ was used to determine a similar screening technique. This results in self-consistent mBJ parameters c_{mBJ} for atoms, which (once determined) were tabulated for use in the initial atom-calculations in FPLO (for stability). Of course the basis depends on the choice of c_{mBJ} . All this is a consequence of the construction of mBJ and some of these conceptual issues are already mentioned in the original publications. Especially, the fact that mBJ is defined for extended solids and has problems for finite systems has an influence on the implementation into FPLO, which for each compound depends on solving the atom first.

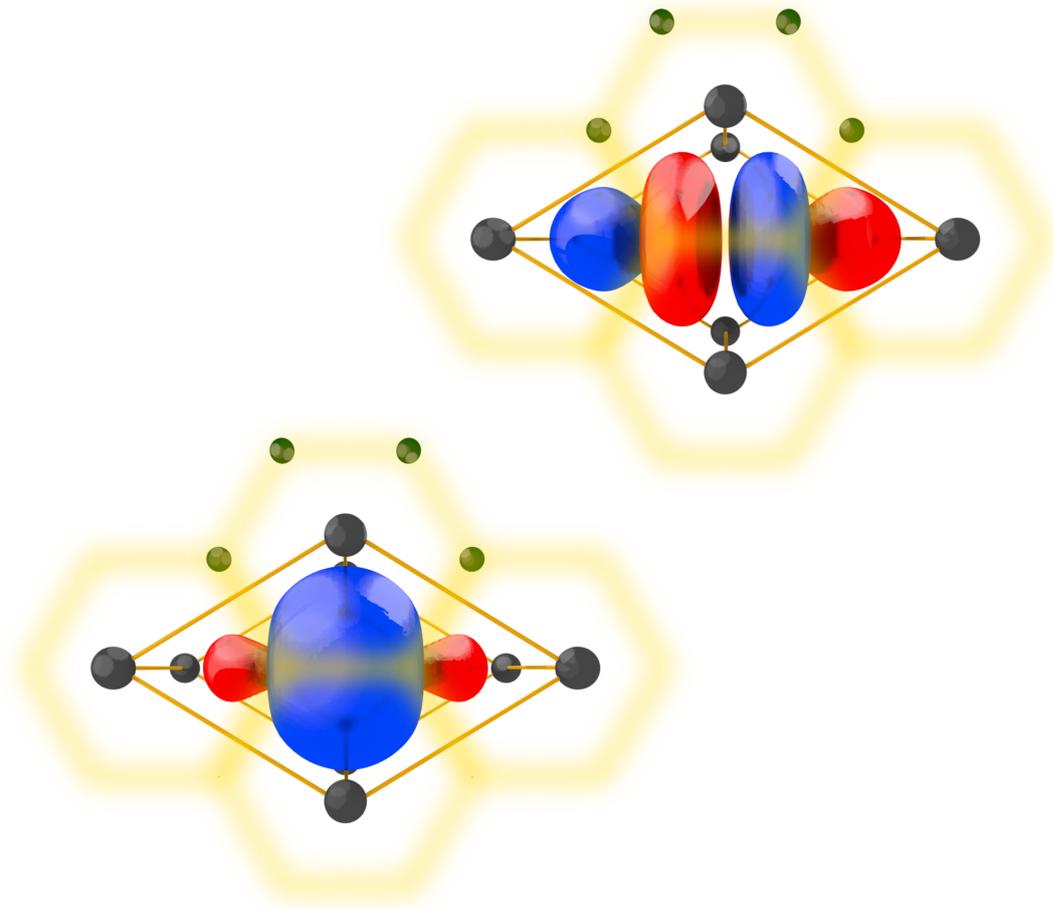
Because of the conceptual issues with applying BJ06-type potentials to molecules currently these potentials are disabled in molecule mode. The interested user can un-comment the relevant lines in `fplostartup.f90` to circumvent this prohibition. However, it is almost certain that some intermediate functions internal to the workings of FPLO will look quite bad.

As a curious finding we have found that for anti-ferromagnetic FeO the self-consistent solution using one of these xc-potentials results in a ground state, which differs from that reported by WIEN. It is a valid solution, just one without a gap. By using an LSDA+U calculation to force the desired order of irreps and produce a starting density we were able to obtain the gapped solution.

 Chapter 6

Wannier functions with FPLO

co-author Marc Höppner



6.1 Intro

Wannier functions can be defined in many ways since there is a gauge freedom of choosing a phase transformation. One way of fixing the gauge is the requirement of maximum localization [7] (ignoring the constraints of the point group). This is a tedious algorithm and even more complicated, if one adds point group symmetry [9, 13]. We do not do this in FPLO. However, it turns out that our definition [3, 5] leads to highly localized Wannier functions (WFs) which obey the symmetry relations of the original crystal by construction. If the WFs turn out to be not well-localized, then in most cases the WF is badly chosen. The main drawback of our approach is that the user has to decide, where the WFs shall sit and which symmetry they shall have. However, this is at the same time intended, since for modeling it is exactly what one wants to do.

The WF sitting in cell R and being of type μ (which denotes the WF center and its symmetry) is defined as

$$W_{R\mu} = \int e^{-ikR} \sum_n \Psi_n^k U_{n\mu}^k \quad (6.1)$$

where Ψ denotes the Kohns-Sham (KS) functions and U is a unitary matrix. If we define less WFs than KS functions, U is a column unitary projector ($U^+U = 1$, $UU^+ = P$). It maps all the KS bands on some few WFs. The choice of U is the choice of the gauge. In FPLO we use a chemically motivated local orbital basis Φ to construct KS functions

$$\Psi_n^k = \frac{1}{\sqrt{N}} \sum_{Rsv} \Phi_{Rsv} e^{ik(R+s)} C_{sv,n}^k$$

(s is the atom position and ν some quantum numbers specifying the orbital). These orbitals although non-orthogonal are in a way 'optimally' localized by their construction. Hence, it is clear that a WF centered at an atom and having a certain orbital symmetry has the corresponding orbital as its main contribution. This allows the following choice of U . We project the KS functions on a test function χ , which is an FPLO orbital in the simplest case. The resulting number is the square root of the orbital character of the KS bands, as plotted in the FAT bands. If we now select only the KS functions with a large such orbital character in Eq. (6.1), we will end up with a WF resembling χ the most and this is where the localization of our WFs comes from. If we want WFs corresponding to sub bands of a band complex with a character χ we have to project onto a particular energy window as well. This happens if there are bonding (B) and anti-bonding (AB) bands of character χ and if one wants, say, only WFs of the AB bands. So, in total we define a test function or WF projector χ and an energy window for each WF. The WF projector χ can in principle be a linear combination of FPLO orbitals.

Example: suppose we have a Cuprate plane. The bands are formed by a linear combinations of Cu $3d$ and ligand O $2p$ orbitals. Around certain k -points, the upper bands are clearly AB and hence correspond to a certain molecular orbital with a certain phase relation between the central $3d$ orbital and the O $2p$ orbitals. The lower (B) bands are clearly formed of the same orbitals, but with a different phase relation. If this is the case, then defining WF-projectors using the AB molecular orbital as χ will automatically yield the anti-bonding bands, given that the band topology is dominated by the clear character separation around the considered k -point. This would make the energy window obsolete.

The Wannier transformation can be described in two steps. In the first step the unitary projector is build from the users definitions of χ and the energy windows. U is applied to the KS functions yielding the Bloch sums of the WFs

$$\begin{aligned} W_{\mu}^k(r) &= \sum_n \Psi_n^k(r) U_{n\mu}^k \\ &= \frac{1}{\sqrt{N}} \sum_R e^{ikR} W_{R\mu}(r) \end{aligned} \quad (6.2)$$

$$W_{R\mu}(r) = \int e^{-ikR} W_{\mu}^k(r) dk \quad (6.3)$$

(Note, that the k -integration contains appropriate normalization factors, left out in the formulas or being hidden in the integral measure.) The orbital Bloch functions are

$$\Phi_{s\mu}^k = \frac{1}{\sqrt{N}} \sum_R \Phi_{Rs\mu} e^{ik(R+s)}$$

where the additional phase factor e^{iks} makes live easier by removing the dependence on the coordinate origin. From the formulas we get the representation of the Hamiltonian in orbital Bloch sums

$$\begin{aligned} \underline{H}_{s's}^k &= \langle \Phi_{s'}^k \hat{H} \Phi_s^k \rangle \\ &= \frac{1}{N} \sum_{R'R} \langle \Phi \hat{H} \Phi \rangle_{R's'Rs} e^{ik(R+s-R'-s')} \\ &= \sum_R \langle \Phi \hat{H} \Phi \rangle_{0s',Rs} e^{ik(R+s-s')} \end{aligned}$$

from which we get the KS eigenvalues

$$\varepsilon_n^k = \left(C^{k+} \underline{H}^k C^k \right)_n \quad (6.4)$$

Using the result above, the WF Bloch representation is

$$\begin{aligned} \langle W_{\mu'}^q \hat{H} W_{\mu}^k \rangle &= \sum_n U_{n\mu'}^{q*} \varepsilon_n^k U_{n\mu}^k \delta_{qk} \\ &= \frac{1}{N} \sum_{RP} \langle W_{P\mu'} \hat{H} W_{R\mu} \rangle e^{ik(R-P)} \delta_{qk} \\ &= \sum_R \langle W_{0\mu'} \hat{H} W_{R\mu} \rangle e^{ikR} \delta_{qk} \end{aligned} \quad (6.5)$$

with

$$\begin{aligned} \varepsilon_{0\mu',R\mu} &= \langle W_{0\mu'} \hat{H} W_{R\mu} \rangle \\ &= \int e^{-ikR} \langle W_{\mu'}^k \hat{H} W_{\mu}^k \rangle dk \end{aligned} \quad (6.6)$$

which is the WF Hamiltonian in real space representation, which usually contains the model we are interested in. Its k -representation (Bloch sums) Eq. (6.5) can be diagonalized and will give the bandstructure corresponding to the model. If the WFs represent the whole Hilbert space spanned by all Ψ_n^k of course the resulting WF-bandstructure coincides with the original bandstructure Eq. (6.4). There is a modification one can do, which consists of restricting the matrix elements Eq. (6.6) by removing hoppings with distances above a certain cutoff radius or hoppings, which are smaller than a certain threshold. The resulting restricted hopping matrix can be Bloch summed and diagonalized again. Note, however, that this modified Hamiltonian does not strictly correspond to the WFs calculated above.

The whole transformation can be summed up, by defining a transformation from the FPLO basis into the WF basis

$$\begin{aligned} W_{R\mu}(r) &= W_{0\mu}(r - R) \\ W_{0\mu}(r) &= \sum_{R's\nu} \Phi_{R's\nu} D_{R's\nu,\mu} \end{aligned} \quad (6.7)$$

6.2 The FPLO WF module

The Wannier function module in FPLO is currently a postprocessing tool. First, one needs a converged calculation. Then, the relevant information must be written to the hard disk in order to access it conveniently later. This might take plenty of disk space (see Sec. 6.2.6)! This information is read afterwards and used in a subsequent FPLO run to calculate the desired Wannier functions. In order to use the module a particular file (`=.wandef`) must be created by the user. Since FPLO15 a python script can be used to create default Wannier function definitions for a large set of situations (`../pyfplo/pyfplo.pdf`). If `=.wandef` is found by a running FPLO process the Wannier module is activated, if the keyword `doit` is found on a single line in the file. To disable the module change the keyword into something like e.g. `xdoit`. If FPLO finds the file `=.wandef` with the keyword it will start dumping data after every Kohn-Sham diagonalization process. If the FPLO process comes to convergence (best to start with a converged calculation to begin with) it stops like in normal mode. The file created after the initial data-dumping is `+wancoeff` (besides all the usual files). *Note:* the band structure plot in the `fedit` submenu should be switched on.

On restart of FPLO (provided that `=.wandef` exists and the `doit`-keyword is set therein) all the data are used and the actual WF-module is executed. It reads the WF definitions from `=.wandef` and constructs the WFs accordingly. The WFs can be produced in real space for visualization and the (interpolated) WF Hamiltonian in Bloch representation Eq. (6.5) can be written to the file `+hamongrid` on a (much denser) k -space grid. The WF hopping integrals Eq. (6.6) are produced in the output and are written in a convenient format into the file `+hamdata`, which is used e.g. by the `pyfplo.slabify` module (also see `../pyfplo/pyfplo.pdf`). Be aware that the WF Hamiltonian on the k -space grid is restricted by the cutoff of the real-space Hamiltonian Eq. (6.6) according to user input.

Example: We will walk through the example of CaCuO_2 step by step. The input-files are provided in the example directory. The calculation is done in the ferromagnetic phase of CaCuO_2 (just to have a more complex situation). The goal is to create a WF for the anti-bonding $3d_{x^2-y^2}$ band. In Figure 6.1 we show the spin-polarized

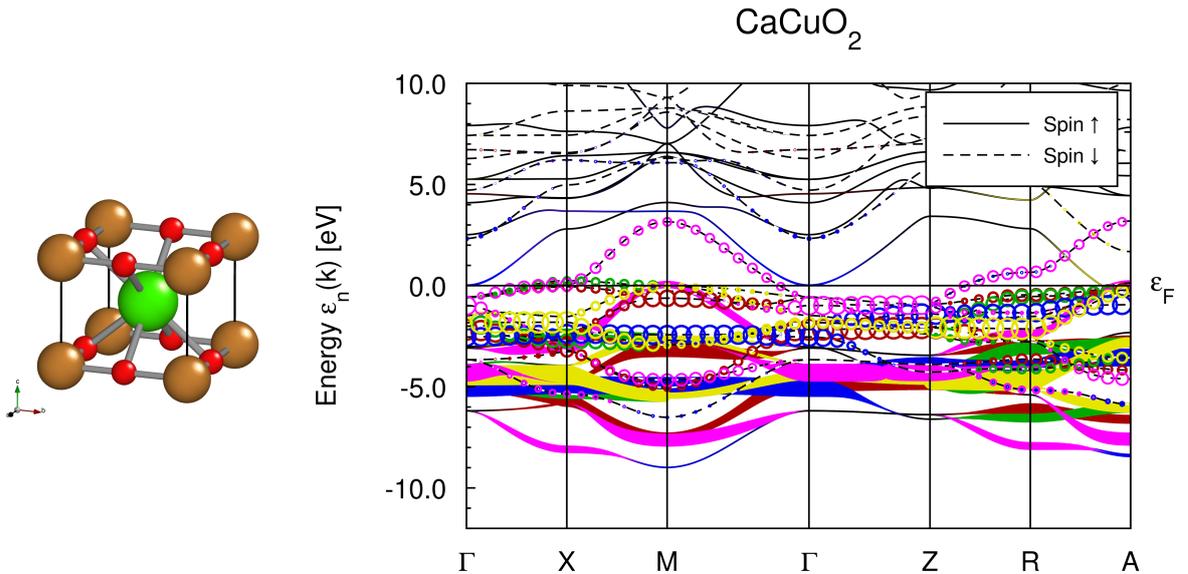


Figure 6.1: FPLO fat-bands CaCuO_2

fat-bands as given by FPLO. The filled symbols denote the $3d$ majority fat-bands and the open symbols the $3d$ minority bands. The $3d_{x^2-y^2}$ bands are called $3d+2$ according to the standard convention and orientation of the global coordinate system and have magenta color. Focusing on the majority bands (filled symbols), you see a bonding and anti-bonding band (best seen at the M-point). We are interested in a description of the anti-bonding band. There is only one Cu atom in the unit cell and according to the list of sites in the output file it is site number 2. We now guess that the Cu, site 2, $3d_{x^2-y^2}$ orbital will have the largest contribution to a WF describing the anti-bonding part of the filled magenta band(s). Thus, we choose this orbital as a WF-projector χ . We assume that the file `=.wandef` exists and that the keyword `doit` is set and that the file `+wancoeff` got already created.

We have to define, which linear combinations of local orbitals are used as WF-projectors. Note that we have two basic choices: the WF-projector *and* the energy window. In our case of CaCuO_2 we already identified the projector. The simplest projector consists of one single orbital centered on site 2. A WF-projector is defined using the keyword `wandef`. A `wandef` can have many contributions `contrib`, each denoting an FPLO orbital with a certain weight factor. Here comes the example

```
...
----- Cu -----
wandef
on
  name Cu x2-y2
  emin -4 -1
  emax -1 3
  de 1 1
  contrib
    site 2
    difvec 0 0 0
    xaxis 1 0 0
    zaxis 0 0 1
    orb 3d+2
    fac 1
```

As can be seen there can be additional lines in the file, and they are considered comments as long as they do not start with a keyword. In our case `'----- Cu -----'` is such a comment. Subsequently, the keyword `wandef` starts a WF-projector section. The next keyword is `on` or `off`. This allows to have several `wandefs` in one file, where only some are used (playing around). The `wandef` has a name, which can be anything after the keyword `name` (actually it should not be more than 17 characters). The energy window is discussed later. In our example the `wandef` is made of one `contrib`. The `contrib` is an orbital sitting at `site 2`, being `difvec 0 0 0` away from the (imagined) center point of the WF, which in this case is the position of site 2. The orbital is defined with respect to the local coordinate system, whose `xaxis` is `1 0 0` and `zaxis` is `0 0 1`. In this local system the `orb` is `3d+2` and the `contrib` enters the whole `wandef` with relative weight `fac 1`. If there are more than one `contribs` the `fac` of each `contrib` can be given. The whole thing is normalized automatically. If the `wandef` is made of several `contribs` at different sites it is important to correctly define the difference vectors `difvec`. Choose a particular point in space as WF center and express the sites of the `contribs` by vectors pointing from the WF center to the sites ... these are the `difvecs`.

If there is more than one site belonging to a crystallographic orbit (all sites generated by the same Wyckoff position), the user has to give every WF-projector separately in the current implementation. In this case it is important that the user takes care of all the different symmetry-related WF-projectors belonging to the same cryst. orbit. This means setting up proper `difvec`, `x/zaxis`, `orb`, `fac` and so on values.

In our example the orbital used as WF-projector forms a bonding and an anti-bonding part. There is only one

Cu $3d_{x^2-y^2}$ orbital in the simple unit cell and hence it can only form one band. However, this orbital forms sigma-bonding hybrids with the plaquette oxygen $2p$ orbitals and this creates “two” bands having a shared character of Cu $3d_{x^2-y^2}$ and O $2p$. Moreover, it is clear from Figure 6.1 that the anti-bonding (AB) band is not isolated due to other hybridizations. Hence, a well defined Wannier function cannot exist. In order to have some approximation for modeling we create a single WF of AB type. According to the rules of how WFs are created it is evident that we need to project away the part of the KS spectrum, which contains the bonding part of the corresponding band. This is done by specifying an energy window. All bands outside this window will be ignored. For the majority bands we have the AB band between -5 and 0 eV and the B band between -8 and -5 eV. For the minority band it is shifted upwards accordingly. The energy window could be defined as a sharp cutoff. This can lead to weird effects, if the band character specified through the WF-projector occurs ONLY outside this window for a particular k-point (such things can actually happen). This would mean that for one k-point the weight of the WF-projector in the considered Hilbert-subspace is zero and this leads to an indefinite problem, when calculating the WFs. So, it is better to make the window smooth in order to have the interesting subspace in the main energy window but allowing to sample outside of it, in case we have a character-run-away scenario. The energy window is defined by a function being 1 between **emin** and **emax** and falling off as a Gaussian with a width **de** outside this window. Of course, the energy window applies to all **contribs** of one **wandef**. In spin-polarized cases these three keywords take two values, one for the majority and one for the minority bands. The user should also pay attention that the energy window has to be the same for all **wandefs** belonging to the same **cryst. orbit**!

Suppose that we defined the above described WF projector onto one orbital, having one WF per unit cell and that we did not specify the energy window. This would lead to a WF whose corresponding band will be pretty dispersionless situated at an energy between the B and AB bands in the original band structure. This happens because the projector χ makes one single band out of the B and AB part of the KS-Hilbert space, which essentially forms two bands. In mathematical terms this will lead to an average of the B and AB KS wavefunctions, which will have non-bonding character. That should also make clear what happens, if we extend the energy window more and more towards the B energies. This will mix in more and more of the bonding KS functions, which leads to a WF band whose energies get pulled down more and more. The user is encouraged to play with these values to get a feeling for the construction of specific WFs.

Now, we have defined all the stuff in the file `=.wandef` and we can run FPLO.

1. The first thing in the WF module will be that the content of `=.wandef` is copied to the output, followed by a more condensed printout of the Wannier parameters. The latter output shows all parameters, also the ones not explicitly set in `=.wandef` (which will have their default values). Then follows a section, in which the symmetry of the WF-projectors χ is checked. If this check does not run through properly there is a mistake in the symmetry relation between **wandefs** of χ -s belonging to the same **cryst. orbit**. Check all keywords. In the moment the energy window is not checked for proper symmetry setting. So, if the code runs through, but the output seems weird, check the energy windows!
2. Now, if the file `+wancoeff` is found it will be loaded. If not, the normal FPLO execution will continue. Reading the data is not the fastest that is why there is an internal loop (see below). See also Sec. 6.2.6.
3. Now, the Wannier bands get processed. As a result the hopping matrix elements between WFs are printed to the output. This printout is controlled by user defined restrictions to avoid enormous amounts of data. Example:

```
spin 1: WF(Cu x2-y2) -> WF(Cu x2-y2) at relative
T=  0.00000  0.00000  0.00000  hop= -2.093458893797475
T=  0.00000  0.00000  6.04712  hop= -0.061341124468540
T=  0.00000  0.00000 -6.04712  hop= -0.061341124468540
```

```

T= -7.29434  0.00000  0.00000  hop= -0.463606088774015
T=  0.00000 -7.29434  0.00000  hop= -0.463606088774014
T=  0.00000  7.29434  0.00000  hop= -0.463606088774014
T=  7.29434  0.00000  0.00000  hop= -0.463606088774015

```

The WFs are given by their name, and the symbol '->' means that the vectors, which follow ('T= ...') point from the WF left of '->' to the one right of '->'. After each vector 'T=' the hopping element in eV is printed. If T=0 the hopping element is the onsite matrix element. The information written to the output is restricted such that only hoppings with $|t| > \mathbf{WF_ham_threshold}$ are considered and only for $|T| \leq \mathbf{ham_cutoff}$.

4. After this the output on the reciprocal grid is performed. It is important to realize that the Hamiltonian in k -space is constructed from the truncated real space Hamiltonian, i. e. the Bloch sums of the hoppings Eq. (6.6) after cutting off of the hoppings for which $|T| > \mathbf{ham_cutoff}$ and $|t| < \mathbf{WF_ham_threshold}$. The resulting Hamiltonian is written to `+hamongrid`. See the `wannier.f90` source code for the order of the data.
5. The very same hopping data are also written to `+hamdata` for further processing by `pyfplo`.
6. Now, `+WF_coefficients` is written, which contains information about the contributions of the FPLO orbitals to the WFs Eq. (6.7). This file shows only contributions, which are larger than `WF_coeff_threshold` in `=.wandef`.
7. At the end of the module the WF are written to disk on a real space grid into files `wfdata...`. Those can be used to visualize the WFs using XFPLO (also see XFPLO help screens of the WF/Gridplot dialog in [../Xfplo/xfplo.pdf](#)):

```
xfplo =.in wfdata001 wfdata005
```

or if the `use-data-directories` option (in FEDIT) was used

```
xfplo =.in +wfdata/wfdata001 +wfdata/wfdata005
```

8. After this FPLO pauses with the message

```
CTRL_C for abort, enter for next trial
```

Here, one can type CTRL-C to stop, or one modifies the `wandefs` (in another window/editor) and hits enter to re-run the module, with re-scanning `=.wandef` but without re-reading `+wancoeff` (which is slow, since it is a large file).

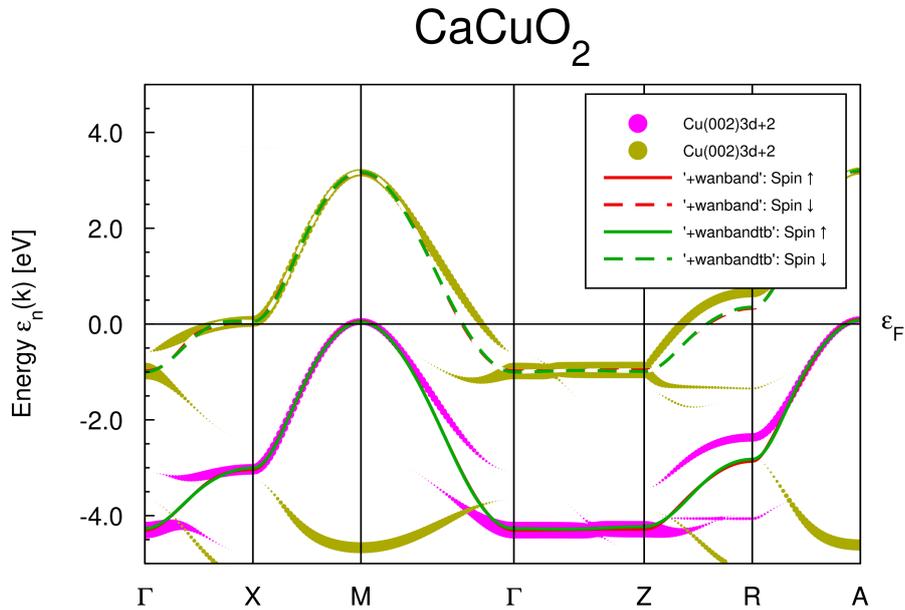
Figure 6.2: Wannier fit for CaCuO_2

Figure 6.2 shows the result of the CaCuO_2 example. Dark yellow and magenta are the $3d_{x^2-y^2}$ fat bands as produced by FPLO. Red shows the full Wannier function transformation (without cutoffs) from the file `+wanband` and green shows the model WF bands including cutoffs from the file `+wanbandtb`. One can nicely observe how the WF bands interpolate the hybridization gap between Γ and X, and M and Γ .

6.2.1 Files

There are three band structure files produced.

+wanband The Bloch-sums of the WF Eq. (6.2) are obtained by doing the unitary transformation in k -space (U). The Hamiltonian in WF-Bloch basis in k -space Eq. (6.5) is directly related to the WF-Bloch sums and hence is obtained straight from the U -transformed KS functions and Hamiltonian. The resulting k -dependent Hamiltonian has the dimension of the WF-basis defined by the user. It can be diagonalized to get the band structure belonging to the WF model. The result of this is written to `+wanband`. If the WFs describe an isolated band complex (one needs as many WFs as there are KS bands in the band complex) then the `+wanband` band structure must coincide with the bandstructure of this band complex in the full FPLO band structure plot. Deviations are possible, if there are not enough k -points in the FPLO-SCF calculation. This comes about since a discrete approximation to the k -integral in Eq. (6.1) defines WFs, which have periodic images with a period of the Born von Karman torus.

+wanbweights This file contains the fat-bands corresponding to `+wanband`. The band weights are determined with respect to the WF character in the corresponding bands. Let's make that clearer. In a local orbital basis like FPLO the overlap of the orbitals with the KS functions determines the fat-bands or how much of a certain orbital a band is made of. This concept can be applied to the WFs as well thinking of the WFs as a basis. So, how much of a particular WF makes up a band? This is given by the information

in this file. Note, that the FPLO fat-bands and WF fat-bands do not coincide. That is the main reason why we do a WF analysis, see Figure 6.3

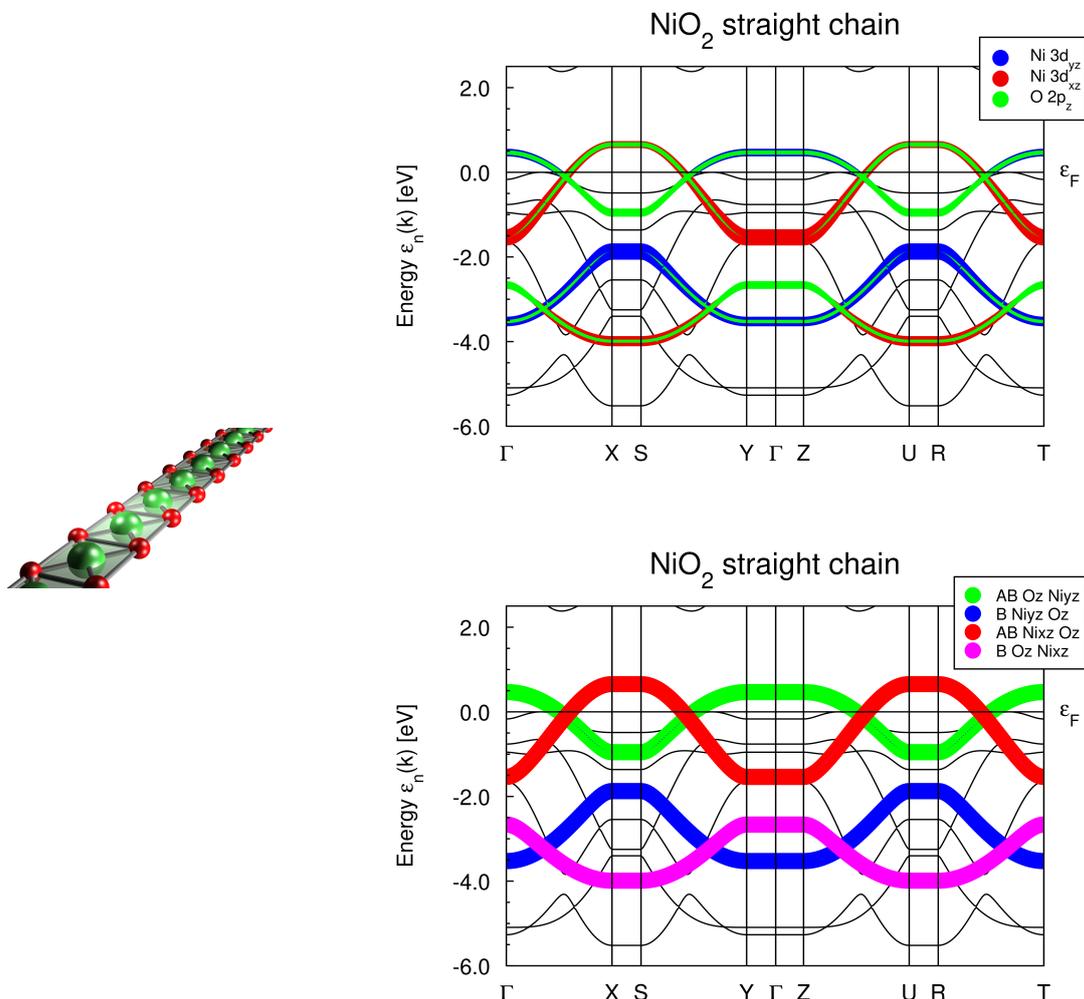


Figure 6.3: Comparison of FPLO fat-bands (upper panel) with WF fat-bands (lower panel) for a NiO_2 -chain. Some WFs are linear combination of several orbitals, such that the WFs are centered between two orbitals. One can clearly see that in the WF basis each band has a pure character, i.e. the bands are completely decoupled by choosing a symmetry adapted basis (WFs).

Now, we can go a step further and Fourier transform the WF-Bloch sums Eq. (6.2) into real space Eq. (6.3). This provides the actual WFs. In real space these WFs overlap (ensuring orthogonality) and the expectation value of the Hamiltonian in this basis forms the hopping integrals Eq. (6.6). These hopping integrals can be cut off at some distance or if they are smaller than a threshold. This defines a modified Hamiltonian in real space. We transform this modified (model) Hamiltonian back into k -space, diagonalize it and get the band structure in this file. If the cut off and threshold is moderate the result should equal `wanband` otherwise it can differ. Especially, for non-isolated bands we can get non-analytic behavior in `wanband`, which is unavoidable in some cases. It turns out that the real-space cutoff reduces these unwanted effects.

The band structure data files can be visualized using the bandplot-tools of FPLO. Besides these files there are a few more. If the use **data directories flag** in `fedit` is set most of these files are written into subdirectories for sake of keeping the directory clean. Exceptions from this rule are `+hamdata` and the old-style `opendx` interface files.

`+hamdata...` contains the WF Hamiltonian in real space after the cutoff procedure (see [../pyfplo/pyfplo.pdf](#)).

`wfdata...` contain the real space Wannier function for visualization with the help of XFPLO or `opendx`. These files can be directly loaded into XFPLO. The `opendx` interface is considered to be obsolete and is no longer switched on by default. Use the `opendx_interface` keyword to switch it on.

```
xfplo =.in wfdata001 wfdata005
```

or if the `use-data-directories` option (in FEDIT) was used

```
xfplo =.in +wfdata/wfdata001 +wfdata/wfdata005
```

(also see XFPLO help screens of the WF/Gridplot dialog [../Xfplo/xfplo.pdf](#))

`+WFstat...` contains the absolute values of the coefficients in Eq. (6.7) as a function of the distance of the corresponding orbitals from the Wannier function center. Plot this file (e.g. `xfbp ./+WFstat.xpy`) to have an idea of the localization of the WFs.

`+Rstat...` contains the statistics of the position operator matrix elements. Plot this to estimate the falloff of this operator (e.f. `xfbp ./Rstat.xpy`).

`+T...` contains the hoppings from a given WF to the neighbouring WFs corresponding to Eq. (6.6) as a function of the distance between the WF centers.

`WFstat.xpy` use this to plot the orbital contribution statistics (`+WFstat...`)

```
xfbp WFstat.py
```

`Rstat.xpy` use this to plot the position operator statistics (`+Rstat...`)

```
xfbp Rstat.py
```

`opendxWF.dx`, `WF.net` and `WF.cfg` these are the `opndx` interface files, which are used in the following way

```
dx -image WF.net
```

6.2.2 Keywords

The keywords in `=.wandef` are explained in the following.

doit switch the WF module on.

keeprunning can be **on** or **off**. If this keyword is **on** or if it is absent the program will not terminate, which is useful for a re-run of the WF-module after changes to `=.wandef`. This avoids the loading of `+wancoeff(+wancoeffbin)`. By default the program keeps running. So what you want to use to overwrite this behaviour is **keeprunning off**.

save_spin_info save the spin operator in **+wancoeff** (only in full relativistic mode). This option makes the file bigger and must be set before **+wancoeff** is created!

save_bfield save the exchange field in **+wancoeff** (only in full relativistic mode). This option makes the file bigger and must be set before **+wancoeff** is created!

save_position save the invariant position operator matrix elements in **+wancoeff**. This option makes the file bigger and must be set before **+wancoeff** is created! (see **gradorder**, below)

gradorder can be 1, 3, 5 and 7. In the calculation of the position operator a numerical gradient formula is needed. The position operator converges slowly with the density of SCF k -mesh (by its very nature). The default is **gradorder 1**. In many cases **gradorder 3** and a default mesh-density gives a more accurate result. At least it speeds up convergence with the mesh density. Possible values are 1,3,5,7 but 5 and 7 seem to not improve the result. The current implementation for higher orders is probably not good for strongly anisotropic k -mesh distances.

Note, that if the k -mesh subdivision is 1 in some direction(s) the derivative formula will be dimensionally reduced, assuming that there is no dispersion in this direction. This yields better results than the full formula, but only if the dispersion is really small. It will however, not always yield the same as a full (isotropic) k -mesh, due to the finite nature of the formula. Both the full mesh and the reduced mesh will converge to the same result for a fine enough mesh, though (hopefully).

In general the position operator is very sensitive to the disentangling energy window fall off and the number of SCF k -points. This is so, since it is used in the MAXLOC-WF algorithm to smooth the Hilbert space, which for our maximally projected case must mean strong dependence on the energy window.

wandef start a definition of a single Wannier function.

on/off make the corresponding **wandef** active/inactive.

emin the lower bound of the energy window (two numbers if spin-polarized).

emax the upper bound of the energy window (two numbers if spin-polarized).

de the width of the Gaussian tail above and below **emax/emin** defining a smooth energy window (two numbers if spin-polarized).

delower/deupper like **de**, but for the lower/upper end of the energy interval. Note, that **de** will overwrite **delower/deupper** if specified after the latter!

ubands/lbands band indices, which specify the maximum and minimum band to be contained in the projector (energy window). (two numbers if spin-polarized)

contrib add an FPLO orbital to the **wandef**. There can be several **contribs** in one **wandef**, one after the other.

site the site number of the orbital according to the FPLO output.

name an arbitrary name to identify the WF.

difvec the distance of the contributing orbital from the WF center. The **difvec** of the first contrib to a **wandef** together with the **site** of the first **wandef** implicitly define the WF center. The choice of the WF center decides whether the WF is real or not. The WF must be defined such that they are real. E.g. a single orbital/contrib WF should always have **difvec 0 0 0**. (In full-relativistic mode WFs are likely to be complex not matter how the center is chosen.)

xaxis the local x-axis expressed in global coordinates

zaxis the local z-axis expressed in global coordinates

sxaxis/szaxis the local x/z-axis for the spin wave functions expressed in global coordinates for full relativistic mode.

This can be a string or a vector. If **sxaxis** is a vector **szaxis** must also be defined and a vector. If it is a string, **szaxis** need not be and should not be specified.

long	short	meaning
'global'	'glo'	use the global cartesian axes
'local'	'loc'	use the local axes as defined by xaxis/zaxis .
'quant'	'qua'	use the global spin quantization axis as defined in FEDIT

By default (if not specified) the spin axis points into the global spin quantization axis as defined in FEDIT. If you want to construct projectors for full relativistic calculations using hand made linear combinations of orbitals the spin axes matter for the actual orbital shape as well as for transformation properties, especially if local coordinate systems are used. Example: the Osmate $j_{\text{eff}} = \frac{1}{2}$ and $j_{\text{eff}} = \frac{3}{2}$ representations for the t_{2g} subspace contain functions like

$$\left| j_{\text{eff}} = \frac{1}{2}, m_{\text{eff}} = \frac{1}{2} \right\rangle = \frac{1}{\sqrt{3}} (|d_{yz} \downarrow\rangle + i |d_{xz} \downarrow\rangle + |d_{xy} \uparrow\rangle)$$

which contain specific orbital-spin combinations. In order to use the proper coordinate system for the spin part we need to specify **sxaxis='loc'**. Then we only need to specify the **xaxis/zaxis** pair to rotate the projector in the local coordinate system, including the spin-part.

The first **contrib** of each **wandef** is written into the **wfdata** files to be available for visualization in XFPLO.

orb the orbital (e.g. 2s+0 or 3d-1). In full-relativistic mode the orbitals can either be pseudo non-relativistic projections denoted by “3d-1 up” or “3d-1 dn” or spherical spinors denoted by 3d3/2-1/2 or 3d5/2+3/2.

fac a weight factor, determining the relative weight of the orbital/contrib in cases of multiple-contrib wandefs. The weights need not to be normalized, this is taken care of.

automode can be **valence**, **all** or **none**: This option makes **wandef** definitions unnecessary. When this mode is not none, **wandefs** for all valence or all semi-core and valence orbitals are created. This leads to a larger Wannier basis and slower calculations. It helps however for automated tasks. Be aware, that a larger **ham_cutoff** might be needed. In this context also a larger SCF k-mesh might be needed, since a too small SCF k-mesh misses Fourier components of the more extended higher lying orbitals. This is demonstrated in DOC/Tutorials/wan. If **automode** is **valence** and if some semi-core bands overlap the valence sector, semi-core orbitals are added to the Wannier basis until the totality of resulting bands has a clear gap below the lowest band. For convenience a python script called **makewandeffromauto.py** is created. To modify this file, please copy it.

coefficients_format can be **bin** or something else. Since version 14.00, the file **+wancoeff** can be converted into binary format for faster loading. After the first FPLO run with a valid **=.wandef** present, **+wancoeff** will have been created. A rerun will start the WF creation process. If this option is set to **bin** the data file will be converted into binary format, if not already done. On any further run the binary file **+wancoeffbin** will be read instead of **+wancoeff**. This is faster. The user has to take care of deleting **+wancoeffbin** whenever **+wancoeff** got changed due to settings changes by the user.

ham_cutoff restricts output of the real-space WF Hamiltonian in standard output and **+T...** Also restricts the matrix elements used in creating **+hamongrid** and **+hamdata**. Since version 19.00 the reach of the matrix elements is restricted by the size of the SC k-mesh (dual R-lattice of the same size). This avoids replica and allows to give a large cutoff for safer results when **automode** is used.

WF_coeff_threshold restricts the output of coefficients in `+WF_coefficients`.

WF_ham_threshold restricts the output of real-space Hamiltonian in standard output and `+T...` and restricts the hoppings used in creating `+wanbandtb`, `+hamongrid` and `+hamdata`.

WF_write_coeff_stats can be **on/off** and triggers the output of the files `+WFstat...`

ham_write_t_stats can be **on/off** and triggers the output of the files `+T...`

print_T can be **on/off** and triggers the printing of the `T=...` lines on standard output ...

do_WF_in_real_space can be **on/off** and triggers the output of `+WFstat` and open-dx files... everything, which is related to the real space representation of the Wannier function. (If it is **off**, the speed increases.)

For the output of the WFs on the real space grid (visualization) we have to define a grid

WF_grid_basis can be **conv/prim**. This defines the basis $\underline{B} = \begin{pmatrix} \vec{b}_1 \\ \vec{b}_2 \\ \vec{b}_3 \end{pmatrix}$ of the box-like grid. We can use the conventional or the primitive basis vectors.

WF_grid_directions in terms of the basis we can define three vectors forming the rows of $\underline{V} = \underline{DB}$. These three directions span the grid-box. The input here is the matrix \underline{D} .

WF_grid_subdivision subdivide the box along the directions \underline{V} accordingly.

WF_grid_origin put the origin of the box here. If this keyword is commented out (e.g. `xWF_grid_origin`) the box will be centered around the WF center.

For the output of the WF Hamiltonian on the k -space grid (file `+hamongrid`) we have to define a grid

k_grid_basis the basis \underline{B} in reciprocal space. (Note the reciprocal relations between bcc and fcc and the like.)

k_grid_directions the directions $\underline{V} = \underline{DB}$ defined here by giving \underline{D} .

k_grid_subdivision subdivisions, see above.

k_grid_incl_periodic_points can be **on/off**. The Hamiltonian in k -space is periodic. We can include or exclude the periodically equivalent points at the boundary of the box.

opendx_interface if this keyword is found the old-style `opendx` interface files (`opendxWF.dx`, `WF.net` and `WF.cfg`) are produced. In the newer version the real space Wannier function data (`wfdata...`) can be loaded into XFPLO directly.

6.2.3 Definition of real spherical harmonics

We define real spherical harmonics Y_{lm} with magnetic qn. numbers $m = -l, \dots, l$ as

$$Y_{lm}(x, y, z) \propto P_l^{|m|} \left(\frac{z}{r} \right) \begin{cases} \sin(|m|\varphi) & m < 0 \\ \cos(|m|\varphi) & m \geq 0 \end{cases}$$

The sin/cos can be expanded according to the addition theorems, e.g. $\sin(2\varphi) = 2\sin\varphi\cos\varphi$. Using $\sin\varphi \propto y$ and $\cos\varphi \propto x$ we get $\sin(2\varphi) \propto xy$. Additionally we need $P_l^{|m|}\left(\frac{z}{r}\right) \propto$ polynomial of degree $l - \underline{m}$ in z . Thus,

$$\begin{aligned} Y_{2,-2} &\propto P_2^2\left(\frac{z}{r}\right) \sin(2\varphi) \propto xy \\ Y_{2,-1} &\propto P_2^1\left(\frac{z}{r}\right) \sin(\varphi) \propto zy \\ Y_{2,0} &\propto P_2^0\left(\frac{z}{r}\right) \cos(0\varphi) \propto z^2 \\ Y_{2,1} &\propto P_2^1\left(\frac{z}{r}\right) \cos(\varphi) \propto zx \\ Y_{2,2} &\propto P_2^2\left(\frac{z}{r}\right) \cos(2\varphi) \propto x^2 - y^2 \end{aligned}$$

For more specific information on the polynomial in z one has to look up the associated Legendre polynomials $P_l^{|m|}(z)$.

6.2.4 Full relativistic spin-part

In full relativistic mode the spin part is important. The real space representation on the grid for visualization with XFPLO is containing the large and small component for spin up and spin down. Usually, you would focus on the large component. So, effectively we have a spinor

$$\begin{pmatrix} g_{\uparrow} \\ g_{\downarrow} \end{pmatrix}$$

which is expressed via eigenfunctions $\chi_{\uparrow} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\chi_{\downarrow} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ of the spin in the global cartesian system.

In the current implementation a global spin quantization z-axis can be defined in FEDIT which results in a colinear approximation for the xc-field pointing in this direction. The XFPLO interface offers to display the spinor components (or density n or spin-density m , colored by the spin polarization $\zeta = \frac{m}{n}$) in a spin basis which corresponds to the xc-field direction. To be specific the Wannier function (the data in the files `wfdata...`) is expressed in global coordinates as

$$W = \underline{\chi}^T \underline{g} = \left(\begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \begin{pmatrix} g_{\uparrow} \\ g_{\downarrow} \end{pmatrix}$$

The xc-field induces a transformation of the form $\underline{B}^T R = (00B_{z'})$, where R is a 3×3 rotation matrix. The matrix R is obtained as $R = (\underline{e}'_x \underline{e}'_y \underline{e}'_z)$, since then we get for a field in \underline{e}'_z -direction $\underline{B} = B_{z'} \underline{e}'_z$

$$\underline{B}^T R = B_{z'} \underline{e}'_z{}^T (\underline{e}'_x \underline{e}'_y \underline{e}'_z) = B_{z'} (001) = (00B_{z'})$$

The corresponding spin eigenfunctions in this frame are expressed by $\underline{\chi}'^T = \underline{\chi}^T D$, where D is a 2×2 spin rotation matrix with the property

$$\begin{aligned} \underline{\sigma}^T R &= D \underline{\sigma} D^+ \\ \underline{\sigma}^T R^T &= D^+ \underline{\sigma} D \end{aligned}$$

where $\underline{\sigma}$ is the vector if Pauli matrices. Let's express the Zeeman term in this spin basis

$$\begin{aligned} E_Z &= \langle \underline{\chi}'^T | \underline{\sigma}^T \underline{B} | \underline{\chi}'^T \rangle \\ &= D^+ \langle \underline{\chi}^T | \underline{\sigma}^T \underline{B} | \underline{\chi}^T \rangle D \\ &= \langle \underline{\chi}^T | D^+ \underline{\sigma}^T \underline{B} D | \underline{\chi}^T \rangle \end{aligned}$$

where the last step is possible because $\underline{\chi}^T$ is the unit matrix.

$$\begin{aligned}
 E_Z &= \langle \underline{\chi}^T | D^+ \underline{\sigma}^T \underline{B} D | \underline{\chi}^T \rangle \\
 &= \langle \underline{\chi}^T | D^+ \underline{\sigma}^T D \underline{B} | \underline{\chi}^T \rangle \\
 &= \langle \underline{\chi}^T | \underline{\sigma}^T R^T \underline{B} | \underline{\chi}^T \rangle \\
 &= \langle \underline{\chi}^T | \underline{\sigma}^T (\underline{B}^T R)^T | \underline{\chi}^T \rangle \\
 &= \langle \underline{\chi}^T | \underline{\sigma}^T (00B_{z'})^T | \underline{\chi}^T \rangle \\
 &= \langle \underline{\chi}^T | \sigma_z B_{z'} | \underline{\chi}^T \rangle
 \end{aligned}$$

which shows that the field in this basis looks like a field in the z' -direction. Now, the Wannier function in the xc-field frame will be

$$W = \underline{\chi}^T D^+ g$$

which has components

$$\underline{g}' = D^+ g$$

in this spin frame. If the spin polarization energy is larger than the spin-orbit coupling this spin frame will show the purest spin character for Wannier functions, which have a spin axis along the global quantization z -axis, as given by the keywords **sxaxis/szaxis**.

6.2.5 Problems

The number of \mathbf{k} -points used in the SCF calculation influences the Wannier function quality. If the \mathbf{k} -mesh is not fine enough, the band structure determined by diagonalizing the WF Bloch Hamiltonian Eq. (6.5) (**+wanband**) will not coincide with the corresponding bands of the full FPLO band structure (**+band**). (Coincidence can of course only happen anyway, if the WFs describe an isolated band complex.)

If the real space Hamiltonian cutoff is smaller than the extend of the WFs the fitted Wannier bandstructure (**+wanbandtb**) will not coincide with the full WF band structure. This may also be due to a bad Wannier definition, which is not localizing the WFs sufficiently (check energy window, use molecular orbitals – several **contribs** per **wandef** – instead of simple orbitals).

There might be unnatural spikes at certain k -points in the WF band structure. This usually means that the energy window is too narrow and that the character of χ is only large outside the window at the corresponding k -points.

If slabs or chains are calculated, there is an artificial periodicity in the irrelevant directions. If not enough SCF \mathbf{k} -points are used in these directions, we get influences of the artificial periodic replica (actually replica are avoided now, but there still might be artifacts). Hence, either the vacuum spacing is very large, which is detrimental for other parts of the code, or one has to use sufficiently many points in the irrelevant directions. However, always first start with subdivision 1 in this direction and only if this fails try more points. (Only some experimenting can tell.) Especially, when the position operator is requested additional points in this direction might be needed. This operator really is badly converging.

6.2.6 Saving memory (and time)

In order to reduce the size of **+wancoeff** the user can set the FEDIT option “restrict bands to window” in the bandplot submenu. Then lower and upper energy bounds in the same submenu defines the bands, which will be written to **+wancoeff** for all data contained in this file.

Use the **coefficients_format** option in **=.wandef** in order to convert **+wancoeff** to binary format.

6.3 Examples

All examples can be found in [../WannierFunctions/](#).

6.3.1 Hexagonal, Graphene, MgB₂: sp²

Here we analyse the Wannier function choices for a hexagonal lattice with essentially two atoms per unit cell. The main issue is to understand the symmetry considerations.

Have a look at the accompanying example directories for MgB₂.

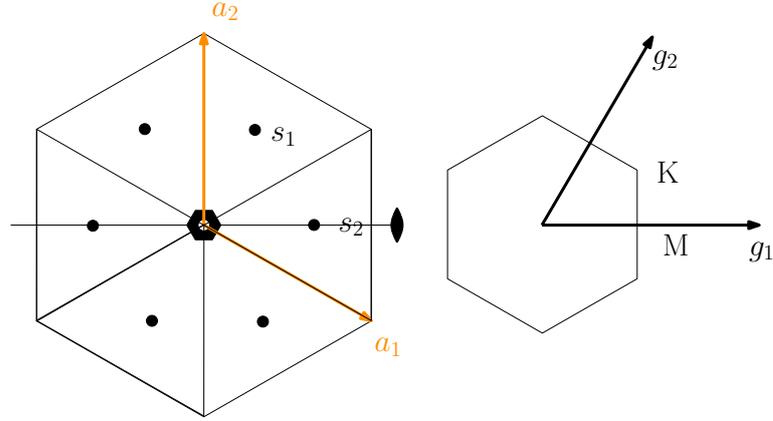


Figure 6.4: Real and reciprocal cell of graphene, MgB₂

Figure 6.4 shows the basic lattice structure. We have a unit cell given by

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{pmatrix} = \underline{\mathbf{A}}\mathbf{E}, \quad \mathbf{E} = \begin{pmatrix} \mathbf{e}_x^T \\ \mathbf{e}_y^T \\ \mathbf{e}_z^T \end{pmatrix}, \quad \underline{\mathbf{A}} = \begin{pmatrix} a_H & 0 & 0 \\ 0 & a_H & 0 \\ 0 & 0 & c_H \end{pmatrix} \begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and two sites $\mathbf{s}^T = \underline{\mathbf{s}}^T \underline{\mathbf{A}}$, $\underline{\mathbf{s}}_1^T = (\frac{1}{3} \frac{2}{3} 0)$, $\underline{\mathbf{s}}_2^T = (\frac{2}{3} \frac{1}{3} 0)$. The reciprocal space is spanned by

$$\mathbf{G} = 2\pi \mathbf{A}^{-T} = \begin{pmatrix} \mathbf{g}_1^T \\ \mathbf{g}_2^T \\ \mathbf{g}_3^T \end{pmatrix} = 2\pi \underline{\mathbf{G}}\mathbf{E}, \quad \underline{\mathbf{G}} = \underline{\mathbf{A}}^{-T} = \begin{pmatrix} \frac{1}{a_H} & 0 & 0 \\ 0 & \frac{1}{a_H} & 0 \\ 0 & 0 & \frac{1}{c_H} \end{pmatrix} \begin{pmatrix} \frac{2}{\sqrt{3}} & 0 & 0 \\ \frac{1}{\sqrt{3}} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

($\underline{\mathbf{A}}^{-T} = (\underline{\mathbf{A}}^T)^{-1} = (\underline{\mathbf{A}}^{-1})^T$) with orthogonality

$$\underline{\mathbf{G}}\underline{\mathbf{A}}^T = \underline{\mathbf{A}}\underline{\mathbf{G}}^T = 1$$

and closure

$$\underline{\mathbf{G}}^T \underline{\mathbf{A}} = \underline{\mathbf{A}}^T \underline{\mathbf{G}} = 1$$

The two high symmetry points in Fig. 6.4 are

$$\mathbf{M} = \begin{pmatrix} 1 \\ 2 \\ 00 \end{pmatrix} \mathbf{G} = \frac{2\pi}{a_H} \begin{pmatrix} 1 \\ \frac{1}{\sqrt{3}} \\ 00 \end{pmatrix}$$

$$\mathbf{K} = \begin{pmatrix} 1 & 1 \\ 3 & 3 \end{pmatrix} \mathbf{G} = \frac{2\pi}{a_H} \begin{pmatrix} 1 & 1 \\ \sqrt{3} & 3 \end{pmatrix}$$

The symmorphic space group is 191 with a maximal point group D_{6h} at (000) with generators $C_6(z)$, $C_2(0)$ and I (indicated in Figure 6.4). Clearly $\hat{C}_6^{-1}\mathbf{s}_1 = \mathbf{s}_2$. The local pointgroup at the two sites is D_{3h} with generators C_3 , $C_2(0)$ and $\bar{6} = IC_6$. For the sake of definiteness let's define operations. We understand the operation g to actively transform the basis vectors $\mathbf{e}_{x,y,z}$ into a new set according to

$$g\mathbf{E}^T = \mathbf{E}^T R(g)$$

where $R(g)$ is the 3D real space representation matrix of operation g . Multiplication from the left with a symmetry operation as in $g\mathbf{E}^T$ denotes the actively transformed object \mathbf{E}^T . If we emphasize that something is the operation (as opposed to the representation matrix) we use an explicit operator symbol as in \hat{C}_6 , while representation matrices are denoted by plain symbols.

For example

$$g = \hat{C}_6, \quad R(C_6) = \begin{pmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

transforms the basis \mathbf{E}^T as

$$\hat{C}_6(\mathbf{e}_x\mathbf{e}_y\mathbf{e}_z) = (\mathbf{e}_x\mathbf{e}_y\mathbf{e}_z) R(C_6) = \left(\frac{\mathbf{e}_x + \sqrt{3}\mathbf{e}_y}{2}, \frac{-\sqrt{3}\mathbf{e}_x + \mathbf{e}_y}{2}, \mathbf{e}_z \right)$$

The coordinates of the actively rotated real space vector $\mathbf{r} = \mathbf{E}^T \underline{r}$ can be obtained from the basis transformation according to

$$\mathbf{r}' = g\mathbf{r} = g\mathbf{E}^T \underline{r} = \mathbf{E}^T (R(g) \underline{r})$$

which can be abbreviated by

$$g\underline{r} = R(g) \underline{r}$$

or $g\underline{r}^T = \underline{r}^T R^T(g)$.

Applied to our example we get the following relation

$$\begin{aligned} \hat{C}_6^{-1}\mathbf{s}_1 &= \mathbf{E}^T C_6^{-1} \underline{\mathbf{s}}_1 \\ &= \mathbf{E}^T C_6^{-1} \underline{\mathbf{A}}^T \begin{pmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 0 \end{pmatrix} \\ &= \mathbf{E}^T (\underline{\mathbf{A}}^T \underline{\mathbf{G}}) C_6^{-1} \underline{\mathbf{A}}^T \begin{pmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 0 \end{pmatrix} \\ &= \mathbf{E}^T \underline{\mathbf{A}}^T (\underline{\mathbf{G}} C_6^{-1} \underline{\mathbf{A}}^T) \begin{pmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 0 \end{pmatrix} \\ &= \mathbf{E}^T \underline{\mathbf{A}}^T \begin{pmatrix} 0 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{3} \\ \frac{2}{3} \\ 0 \end{pmatrix} \\ &= \mathbf{E}^T \underline{\mathbf{A}}^T \begin{pmatrix} \frac{2}{3} \\ \frac{1}{3} \\ 0 \end{pmatrix} \\ &= \mathbf{s}_2 \end{aligned}$$

A set of functions of the coordinates (orbitals) transforms as

$$gf_m(\mathbf{r}) \stackrel{\text{def}}{=} f_m(g^{-1}\mathbf{r}) = f_m(\mathbf{r}^T R(g)) \stackrel{\text{def}}{=} \sum_n f_n(\mathbf{r}) D_{nm}^f(g)$$

where $D_{nm}^f(g)$ are the representation matrices of the group in the space spanned by f_m . In other words the orbitals $gf_m(\mathbf{r})$ are also actively transformed like the Cartesian basis vectors $\mathbf{e}_{x,y,z}$.

In our examples we have a light p element sitting at the two sites of lower than maximal symmetry. The essential basis consists of one s and three p orbitals. Figure 6.5 shows the band structure of MgB_2 , where the boron bands are highlighted. (The Mg bands do not play a big role in this energy window.)

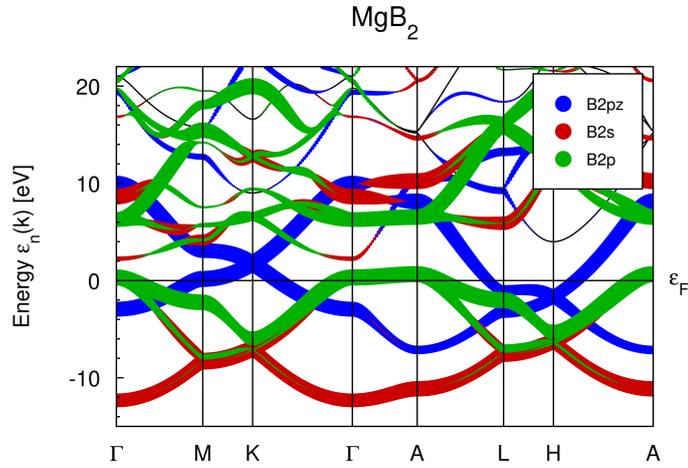


Figure 6.5: Bandweights of boron in MgB_2

The irreducible representations of the site symmetry D_{3h} make $p_{x,y}$ an E'' doublet and p_z a A_2'' singlet, while the s -orbital represents an A_1' singlet. In total we have 8 orbitals and hence bands. Due to the symmetry the p_z bands largely decouple from the other orbitals along high symmetry lines in the $k_z = 0$ and $k_z = \frac{\pi}{c_H}$ planes. At general points they of course couple. Furthermore, these bands cross the Fermi level, while the s and $p_{x,y}$ bands form bonding and anti-bonding band complexes of three bands each separated by a gap. The bonding band complex has exactly three bands, while the AB complex shows a high degree of band entangling with other bands. The two p_z bands cannot be separated into two distinct bands because of their Fermi surface and symmetry (see Sec. 6.3.2). However the planar-orbital bands can be represented by different kinds of Wannier functions.

The simplest way of defining WFs is to use the atomic $s, p_{x,y}$ orbitals sitting at the two sites. This more or less reproduces the FPLO band characters. Another alternative is to try to find WFs for the bonding (anti-bonding) bands only. There are essentially three bands in each band complex and it is immediately clear that atom centered functions cannot fulfill the crystal symmetry due to the number of sites (two). The bands must be combinations of orbitals from both sites and hence an odd (3) number of orbitals from an even number of sites (2) cannot be symmetric. We discussed the irreducible representations of the orbitals in the site symmetry above. We have a singlet (s) and a doublet ($p_{x,y}$) (as is reflected in the degeneracies at the Γ -point) but all three orbitals are mixed in the bonding bands (no further decoupling).

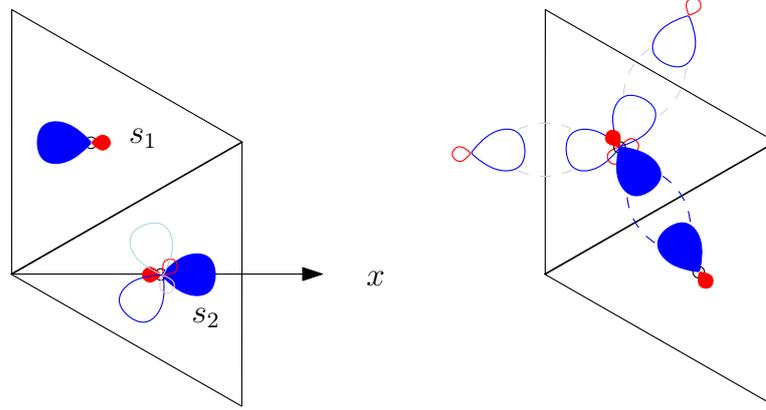


Figure 6.6: sp^2 -hybrid basis. The left panel shows one sp^2 -orbital at both sites (filled) and the remaining two $120^\circ/240^\circ$ -rotated orbitals at s_1 (open). The right panel shows a bonding bond-centered combination (full) and two symmetry related combinations (open).

Unfortunately, there is hardly a simple linear combination of say the $p_{x,y}$ orbitals from both sites, which will fulfill the full crystal symmetry. Hence, we have to resort to reducible representations aiming at constructing as highly symmetric orbitals as possible. The well known answer is of course sp^2 hybrids. We introduce the following orbitals at each site

$$\begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \end{pmatrix} = \frac{1}{\sqrt{3}} \begin{pmatrix} s + \sqrt{2}p_x \\ s + \sqrt{2}C_3p_x \\ s + \sqrt{2}C_3^2p_x \end{pmatrix}$$

which are depicted in the left panel of Figure 6.6. Note, that here C_3p_x is the C_3 rotation of the p_x orbital with the rotation origin at the site not at the global coordinate origin.

Obviously, C_3p_x is not parallel to p_x and hence contains some p_y . On the other hand $(1 + C_3 + C_3^2) = 3P_{\parallel}$ where P_{\parallel} is a projector onto the rotation axis. For in-plane objects $P_{\parallel}p_{x,y} = 0$. This consideration shows that the p -parts of $\Phi_{1,2,3}$ are linearly dependent, however, with the s -admixture they span the same space as s and $p_{x,y}$. The factors are chosen such that $|\Phi_i|^2 = 1$ and that $\sum_i |\Phi_i|^2$ contains one $|s|^2$ density and two $|p|^2$ densities.

Now, we can put three sp^2 hybrids onto each site (rotated by 60° , 180° and 300° at the second site) and clearly they possess maximum crystal symmetry as a reducible 3-dimensional representation. (Figure 6.6, left panel). The symmetry relations are (rotation origin at (000))

$$\begin{aligned} C_6\Phi_{s_2,1} &= \Phi_{s_1,3}, \dots \\ C_3\Phi_{s,1} &= \Phi_{s,2}, \dots \\ C_2(0)\Phi_{s_2,1} &= \Phi_{s_2,1}, \quad C_2(0)\Phi_{s_2,2} = \Phi_{s_2,3}, \dots \\ I\Phi_{s_2,1} &= \Phi_{s_1,2}, \dots \end{aligned}$$

where we have dropped the details which show in which unit cell the transformed orbitals end up. This information is important for a detailed and complete description of the space group symmetry but is not necessary here. What these relations show however is that the sp^2 orbitals at both sites span a basis under the full symmetry. The resulting Wannier function fat bands are shown in Figure 6.7.

The advantage of this high symmetry is that we now can create linear combinations of one sp^2 from each site at the bond centers and by construction this additionally generates two similar symmetry related combinations (Figure 6.6, right panel). This would not be possible if we used the irreducible s and p_{xy} orbitals. In that way

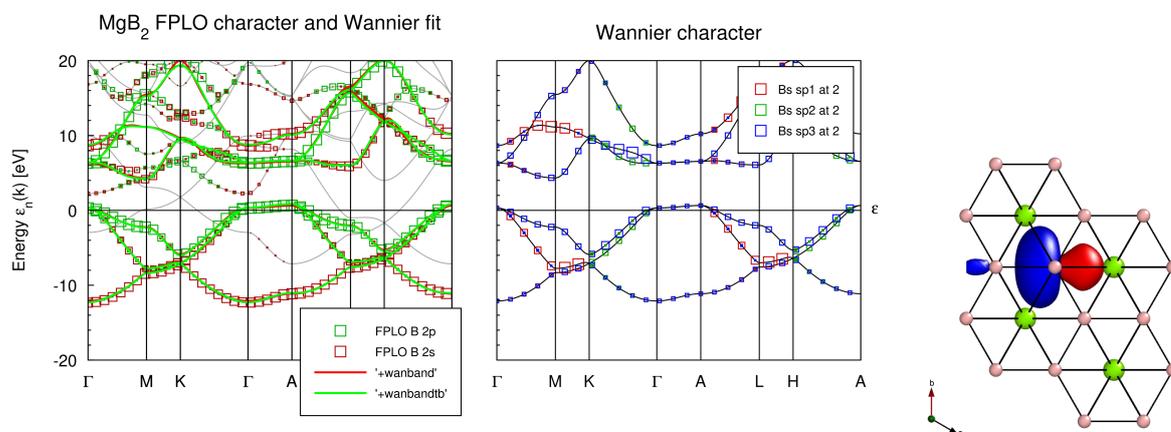


Figure 6.7: Wannier function fit with 3 sp^2 -orbital per site.

one can separate the bonding from the anti-bonding orbitals. This is an example for the fact that in many covalent p -electron systems the bonding/anti-bonding bands must be described by bond-centered Wannier functions (Carbon chain: sp^1 , diamond sp^3). Once we have separated the bonding Wannier functions, we can of course produce either a separate fit for the bonding or anti-bonding bands respectively (Figure 6.8) or a fit for bonding and anti-bonding bands together (however with clear energy separation of the orbital).

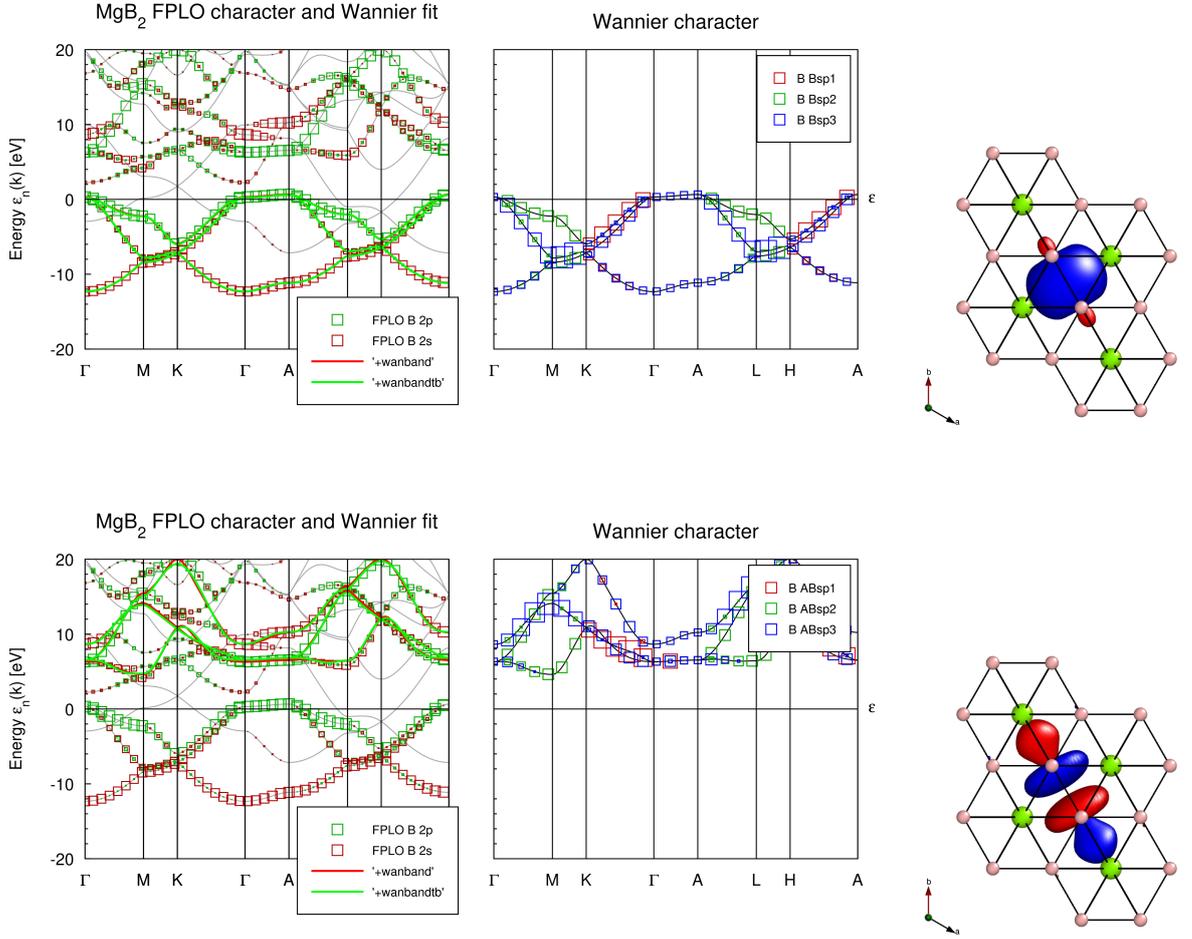


Figure 6.8: Bonding/anti-bonding bond centered Wannier function fits.

One last consideration concerns the choice of energy windows. For the entangled anti-bonding bands to be fitted better we put a narrow energy window onto the occupied bonding bands but allow for a large Gaussian tail at the upper side of the energy window. This pulls the majority weight from the well-separated bonding bands and magically adds the missing anti-bonding weights. If we extend the main energy window to encompass the anti-bonding bands it would pull in weights from higher lying bands, which also show s and $p_{x,y}$ character, which in turn pulls the fitted bands up in energy. This trick can be used rather generally to improve the Wannier fit.

6.3.2 Hexagonal, Graphene, MgB_2 : p_z orbitals

To elucidate the situation with the p_z subsystem we perform a little exercise. As we stated earlier the p_z bands largely decouple from the rest, but have a Fermi surface which prohibits the separation of bonding and antibonding bands

The local p_z basis consists of two functions $\Phi_{R_s}(\mathbf{r}) = \Phi_s(\mathbf{r} - \mathbf{R} - \mathbf{s})$ distinguished by the site label $s = s_1, s_2$. This basis can be used as Wannier projectors and allows the construction of p_z WFs for the p_z two-band complex

as a whole. The corresponding Bloch functions, which form a basis for the eigenstates of the Hamiltonian are

$$\Phi_s^{\mathbf{k}} = \frac{1}{\sqrt{N}} \sum_R e^{i\mathbf{k}(\mathbf{R}+\mathbf{s})} \Phi_{R\mathbf{s}}$$

The convenient additional phase factor $e^{i\mathbf{k}\mathbf{s}}$ is a uniform $U(1)$ phase of each function and does only affect the Boundary condition: $\Phi_s^{\mathbf{k}+\mathbf{G}} = \Phi_s^{\mathbf{k}} e^{i\mathbf{G}\mathbf{s}}$. It removes the origin dependence in matrix elements.

For molecules one can form symmetrized molecular orbitals. So we will introduce them for the lattice as well. The ‘‘benzene-ring’’ orbitals around the point (000) (center point in the left panel of Fig. 6.4) are

$$\Phi_{\pm} = \frac{1}{\sqrt{6}} [(1 + C_3 + C_3^2) \Phi_{s_1} \pm (1 + C_3 + C_3^2) \Phi_{s_2}]$$

Its Bloch functions are given by

$$\begin{aligned} \Phi_{\pm}^{\mathbf{k}} &= \frac{1}{\sqrt{N}} \sum_R e^{i\mathbf{k}\mathbf{R}} \Phi_{\pm} \\ &= \frac{1}{\sqrt{N}} \sum_R e^{i\mathbf{k}\mathbf{R}} \frac{1}{\sqrt{6}} [(1 + C_3 + C_3^2) \Phi_{s_1} \pm (1 + C_3 + C_3^2) \Phi_{s_2}]_{\mathbf{R}} \end{aligned}$$

Note that the subscript \mathbf{R} at the square bracket means that one first constructs the molecular orbital by applying all symmetry operators and then shifts the whole object to the lattice vector \mathbf{R} .

We use the following detailed transformation properties

$$C_3 \underline{s}_1 = -\underline{s}_2 = \left(-\frac{2}{3} - \frac{1}{3}0 \right) = \underline{s}_1 - (110) \quad (6.8)$$

$$C_3^2 \underline{s}_1 = \left(\frac{1}{3} - \frac{1}{3}0 \right) = \underline{s}_1 - (010) \quad (6.9)$$

$$C_3 \underline{s}_2 = \left(-\frac{1}{3} \frac{1}{3}0 \right) = \underline{s}_2 - (100) \quad (6.9)$$

$$C_3^2 \underline{s}_2 = \left(-\frac{1}{3} - \frac{2}{3}0 \right) = \underline{s}_2 - (110) \quad (6.9)$$

to re-express the ring Bloch functions via the local basis Bloch functions (of which we know that they form a basis):

$$\begin{aligned} (1 + C_3 + C_3^2) \Phi_{s_1} &= \Phi_{s_1}(\mathbf{r} - \mathbf{s}_1) + \Phi_{s_1}(\mathbf{r} - \mathbf{s}_1 + (110)\mathbf{A}) + \Phi_{s_1}(\mathbf{r} - \mathbf{s}_1 + (010)\mathbf{A}) \\ [(1 + C_3 + C_3^2) \Phi_{s_1}]_{\mathbf{R}} &= \Phi_{s_1}(\mathbf{r} - \mathbf{R} - \mathbf{s}_1) + \Phi_{s_1}(\mathbf{r} - (\mathbf{R} - (110)\mathbf{A}) - \mathbf{s}_1) \\ &\quad + \Phi_{s_1}(\mathbf{r} - (\mathbf{R} - (010)\mathbf{A}) - \mathbf{s}_1) \end{aligned} \quad (6.8)$$

and similarly for s_2 . Under the Bloch sum we can redefine the \mathbf{R} summation variable to remove the additional lattice vector shifts of Eq. (6.8)

$$\frac{1}{\sqrt{N}} \sum_R e^{i\mathbf{k}\mathbf{R}} [(1 + C_3 + C_3^2) \Phi_{s_1}]_{\mathbf{R}} = \Phi_{s_1}^{\mathbf{k}} e^{-i\mathbf{k}\mathbf{s}_1} + \Phi_{s_1}^{\mathbf{k}} e^{i\mathbf{k}((110)\mathbf{A} - \mathbf{s}_1)} + \Phi_{s_1}^{\mathbf{k}} e^{i\mathbf{k}((010)\mathbf{A} - \mathbf{s}_1)}$$

$$\begin{aligned} \Phi_{\pm}^{\mathbf{k}} &= \frac{1}{\sqrt{N}} \sum_R e^{i\mathbf{k}\mathbf{R}} \frac{1}{\sqrt{6}} [(1 + C_3 + C_3^2) \Phi_{s_1} \pm (1 + C_3 + C_3^2) \Phi_{s_2}]_{\mathbf{R}} \\ &= \frac{1}{\sqrt{6}} \left(\left(1 + e^{i\mathbf{k}(110)\mathbf{A}} + e^{i\mathbf{k}(010)\mathbf{A}} \right) \Phi_{s_1}^{\mathbf{k}} \pm \left(1 + e^{i\mathbf{k}(100)\mathbf{A}} + e^{i\mathbf{k}(110)\mathbf{A}} \right) \Phi_{s_2}^{\mathbf{k}} \right) e^{-i\mathbf{k}\mathbf{s}_1} \end{aligned}$$

One can see that the additional phase factors vanish for $\mathbf{k} = \pm \left(\frac{1}{3}\frac{1}{3}k_z\right) \mathbf{G}$. Hence on the whole line KH

$$\Phi_{\pm}^{\mathbf{k} \in KH} = 0$$

Consequently, the ring functions do not span a basis for the whole BZ. This leads to singularity warnings in the WF module. If, however the input data for the **wandefs** is not precise enough such that numerically the phase factor calculated above is not exactly zero it looks as if the process worked. What you will experience is that the exact WF transform in **+wanband** might fit the FPLO bands while the **+wanbandtb** fit only fits if a very large **ham_cutoff** in real space and a very small threshold **WF_ham_threshold** are considered. This is so because the WFs defined via ring-functions are essentially non-analytic.

In summary, the ring functions are good for molecules but not for extended states. In molecules there is no Fermi surface. The MO levels are discrete and hence in this case the MO WFs work. One last note: the ring functions do not work since the ring combines orbitals of several unit cells with fixed phase factors. There are less (2) resulting MOs than unit cells (3) involved in defining them. This should make clear that a linear combination of the MO Bloch sums cannot have the same flexibility and hence Hilbert space size as the individual orbitals. In truth we left out four of the six possible ring linear combinations. However our Hilbert space has $\dim = 2$ (two bands or p_z orbitals per cell). This too tells us that there is something wrong. A locally complete ring basis has $\dim = 6$ but we use a $\dim = 2$ basis in the unit cell to construct this. The full phase flexibility requires to use all six ring orbitals but the resulting six Bloch bands must be overcomplete.

Of course, in other cases the ring MOs might work, e.g. if the ring is completely contained in a single unit cell and if the other four MOs have higher energy.



Chapter 7

Grep data from output: GREPFPLO

7.1 Specific data

The old grEE flavours of output greping have been enriched/replaced by the program GREPFPLO.

Usually different calculations are performed in different directories, with input values changing from calculation to calculations. If the directory names are given such that they contain the value in the name, output greping is straight forward. Let's suppose we have a bunch of dirs for varying FSM momenta and the dirs are called M=1, M=2, M=2.3 and M=2.5. The output files are called out in all dirs. We can now create a data file containing the FSM momenta and total energies of all four dirs by calling

```
grepfplo -m EE -p M=
```

which results in a file looking something like:

```
1 -141.12
2 -141.13
2.3 -141.14
2.5 -141.13
```

Note, that we changed the table for the onsite orbital momentum to show entries for each site not each sort. This is repetitive information, but more consistent with other tables.

GREPFPLO has a help screen (option -h), which is shown below:

```
Extract data from the fplo output file(s)
```

```
Usage: grepfplo [-h] -m mode [options] (-p prefix) (-f outfile) ([-a(11)])
        ([-x(fbp)]) ([-xm(grace)])
```

```
We assume that separate calculations are done in separate directories
and that the directory names contain the running variable value.
```

```
prefix: is the prefix string of a bunch of directories, of
        which the data shall be extracted. (Default is . =current directory.)
        The prefix is used to select the directories and is removed from the
        directory name to get the corresponding values;
```

E.g. if the directories are named
M=0.1 M=0.2 M=0.3, the prefix M= will result in a data set

```
0.1 data1
0.2 data2
0.3 data3
```

where data... is the data actually extracted from the output files. (See below.)

If the directories are named id=1_M=0.1 id=2_M=0.2 ..., the prefix
'id*M=' (quotes are important) will give the same list as above.

The prefix is used to identify the directories but also
is removed from the directory name to obtain the values of the
first column (which are encoded in the dir-names).

Use single quotes to protect * in shell context;

If prefix is . (simple dot), the file in the current directory is scanned.
In this case the option '-all' will make a list of the data indexed by the
iteration step.

outfile: is the name of the output file, which should be the same in all
directories. Default is out.

modes: Default is EE.

EE: extract the total energy. There is no options. Examples:

```
grepfplo -m EE -p 'id*M=' -f out
```

This extracts all Etots from all directories named id*M=...
where * can be anything and ... after M= is usually the value of M for
this particular directory/calculation
The result is a table of M and Etot values

```
grepfplo -m EE -p M= -f out
```

Same as above, but for directories called M=...

```
grepfplo -m EE -p . -f out
```

Extract last total energy from file out in current directory
Equivalent to grepfplo -m EE or grepfplo

```
grepfplo -m EE -p . -f out -a
```

Extract total energies for all iteration steps from current directory
The resulting table has columns 'iteration step' 'total energy'
Equivalent to grepfplo -a

```
grepfplo -m EE -p . -f out -a -x
```

```
grepfplo -m EE -p . -f out -a -xm
```

Extract total energies for all iteration steps from current directory
and send them to xfbp (-x) or xmgrace (-xm)

This is equivalent to `grepfplo -a -x` or `grepfplo -a -xm`

FF: free energy (if it applies)

EEcorrected: broadening corrected energy, $(E_{\text{free}}+E_{\text{tot}})/2$

SS: total gross spin. no options.

it: iteration progress

fit: force iteration progress

time: timing of one cycle

term: termination of process

SSat: total gross spin of atom. Options: site-number. Examples
`grepfplo -m SSat 13 -p M= -f out`

N_net/N_gros/S_net/S_gros: individual population numbers.

Options: site-number orbital-number.

orbital-number is the number of the orbital in the order as printed in the population analysis.

If orbital-number is out of range the total site population number is printed.

For the N_gros cases there is one more number than for N_net and S_..., which is the number of excess electrons of the site.

Lzat: the orbital moment of an atom. Options: site-number. Example
`grepfplo -m Lzat 12 -p U=`

Bfsm: the auxillary magnetic field needed to set the FSM moment.

dBfsm: the change of the auxillary magnetic field needed to set the FSM moment.

This is for controlling what's going on in full relativistic FSM calculations.

NkTotal: the total number of k-points in the BZ.

spinmoments: table of all spin moments.

orbitalmoments: table of all orbital moments.

gmBJ: the modified Becke Johnson g-parameter.

cmBJ: the modified Becke Johnson coefficient.

gap: the total gap.

7.2 Categorized data

Several modules write their output with a prefix in order to easily extract the data. E.g. the topological insulator module prefixes all output with "TI:". In order to grep it

and to pipe it to a pager or save it to a file you can do something like

```
grep TI: out | less
```

or

```
grep TI: out | more
```

or

```
grep TI: out | tee tidata
```

or

```
grep TI: out > tidata
```

Possible prefixes are listed in Table 7.1.

Module	Prefix	Comments
Forces	FORCES:	
Topological Insulator	TI:	
Optics	OPTICS:	
Molecular fatbands	FATBANDS:	
dHvA iso surface stage	ISO:	
Normal iteration information	SCF:	
Parallel execution	MPI:	not ready yet
Force iteration information	FORCES:	
Preparing basis potentials	vatom[Wyckoff number] (e.g. vatom1)	Only shown, if verbosity level ≥ 4
Start density preparation	startatom[Wyckoff number] (e.g. startatom1)	

Table 7.1: Module output prefixes



Chapter 8

Adding band weights: FADDWEI

Band weights are created if the `bandstructure` plot and `weights` options are set in FEDIT. This will produce `+bweights`. The bandplot submenu of FEDIT contains the option to define a coordinate system for the orbitals to project on (transform axes). In full relativistic mode additionally a projection onto quasi non-relativistic symmetries is provided in the file `+bweightslms`.

The full relativistic basis has quantum numbers $lj\mu$, while the non-relativistic basis has $lm\sigma$ (σ is spin up or down). In not full relativistic mode the spin is encoded by having spin up and spin down bands. This means that for non-spin polarized calculations only one set of bands/weights is needed. In full relativistic mode spin is not a good quantum number, hence a projection into $lm\sigma$ space is approximate. Furthermore, the spin is no longer distinguishing bands. Instead spin becomes a weight itself, which means that the orbitals are now carrying an explicit spin label. For non-spin polarized full relativistic calculations the resulting fatbands are written for both spin directions, for the reason that in systems without inversion symmetry there can be spin polarization of individual eigen functions even though the sum over all eigen functions always produces a non-polarized density.

The weights are normalized such that the sum of the weights of all orbitals at a certain point of the bandstructure add up to one. In full relativistic mode with $lm\sigma$ projection they add up to one summing over the spins, which are now part of the orbitals. If there is an inversion center and a non-spin polarized calculation, Kramers degeneracy will lead to every band being doubly degenerate. In this case the spin-up and spin-down weights will be equal. A pure band with pure orbital character, say Fe $3d_{z^2}$ will actually have weight 0.5 for each spin, so that the weights appear half as small as compared to a not full-relativistic calculation. On the other hand there will be two degenerate bands (inversion center \rightarrow Kramer) which effectively gives the same counting as in not full-relativistic treatment. This has to be kept in mind when interpreting (adding) band weights in such cases.

As an alternative to the default route of producing fatbands the `=.bwdef` mechanism Sec. 9.3 can be used to create custom tailored weights.

Once a file `+bweights...` or another file containing band weights exists it is sometimes desirable to add several weights together to create coarser representations. E.g. the default weights for a $3d$ transition metal would contain weights for each individual orbital $3d_{xy}$, $3d_{yz}$, $3d_{z^2}$, $3d_{xz}$ and $3d_{x^2-y^2}$. The weights file contains a header, which names the character of each weight. In a default file these labels are named e.g. “Fe(003)3d-1”, which stands for iron site 3 orbital $3d_{-1}$. In not-full-relativistic calculations the angular part of the orbitals are defined as real spherical harmonics Y_{lm} with respect to the global cartesian coordinate system. In this system we get the labeling Table 8.1.

In a full relativistic calculations the angular parts of orbitals are defined as spherical spinors.

$$\chi_{\kappa\mu} = \chi_{lj\mu} = \sum_{s=-1}^1 c_{\kappa\mu}^s \chi_s \mathcal{Y}_{l,\mu-\frac{s}{2}}$$

with Clebsch-Gordon coefficients $c_{\kappa\mu}^s$ the complex spherical harmonics \mathcal{Y}_{lm} and the spin-orbit quantum number κ : $\kappa(l, j = l - \frac{1}{2}) = l$, $\kappa(l, j = l + \frac{1}{2}) = -l - 1$. These spinors are completely specified by the quantum numbers $l, j = l \pm \frac{1}{2}$ and $\mu = -j, \dots, j$. The indices $lj\mu$ span a space of $2(2l + 1)$ orbitals, which is the same number as for the non-relativistic lm orbitals if spin is counted ($lm\sigma$).

m	real Y_{lm}				complex \mathcal{Y}_{lm}			
	s	p	d	f	s	p	d	f
-3				$y(3x^2 - y^2)$				$e^{-i3\varphi}$
-2			xy	xyz			$e^{-i2\varphi}$	$ze^{-i2\varphi}$
-1		y	yz	$y(5z^2 - 1)$	$e^{-i\varphi}$		$ze^{-i\varphi}$	$(5z^2 - 1)e^{-i\varphi}$
+0	1	z	$3z^2 - 1$	$z(5z^2 - 3)$	1	z	$3z^2 - 1$	$z(5z^2 - 3)$
+1		x	xz	$x(5z^2 - 1)$		$e^{i\varphi}$	$ze^{i\varphi}$	$(5z^2 - 1)e^{i\varphi}$
+2			$x^2 - y^2$	$(x^2 - y^2)z$			$e^{i2\varphi}$	$ze^{i2\varphi}$
+3				$x(x^2 - 3y^2)$				$e^{i3\varphi}$

Table 8.1: Mapping spherical harmonics to functions. (**Normalization/Phases not included!**)

In a user manipulated file these labels can contain anything. They are however restricted to 17 characters. Now, it could be good to show fat bands for the whole $3d$ set of orbitals, which means that we have to add $w_{\text{sum}} = \sum_{m=-2}^2 w_{3d_m}$. More generally, one could add all weights corresponding to a certain site or several sites and so on. This is achieved by FADDWEI. It supercedes the old programs `addweig` and `addweights`. The program has commentline flags (try `-h`).

FADDWEI reads one weight file (`+bweights...` or such) and one state-definition file. The default is `=.addwei`. When starting fresh it offers to create an example state-definition file. This file tells which weights/orbitals have to be added into a new weight. There are two options to do that. Either specify a complete label for each weight to be added, or specify an element a number of sites and a number of orbitals. Optionally, a factor can be given, which is multiplied to each weight of the corresponding specification before adding all up. This is most likely seldomly necessary.

If pseudo-nonrelativistic projections onto Y_{lm} symmetries are created in full relativistic mode $lm\sigma$ -projection or using the `=.bwdef` mechanism (Sec. 9.3) the default orbital labels contain an additional spin indicator (up/dn) at the end, in which case the input in `=.addwei` must specify the spin. In ordinary calculations the spin is encoded by having one band for each spin direction. This however is not possible in full-relativistic mode, hence the explicit spin in the default labels.

To obtain a complete (long) list of all labels use FADDWEI `-p`.

Examples:

- We have Fe at sites 2,3,4 and 8, oxygen at sites 12,13,14,15,16 and some other stuff. We define one weight with name Fe3d one with “O2p” and one with “Fe O all”.

```
# all Fe 3d
name "Fe 3d"
  atom Fe sites 2..4 , 8 orbitals 3d # all Fe 3d orbitals for sites 2,3,4 and 8
# all O 2p
name='O 2p'
  atom= 0 sites =12..16 orbitals= 2p # all O 2p orbitals for sites 12 through 16
# one weight sum containing Fe and O
name 'Fe O all'
  atom Fe sites 2..4 8 orbitals 3d
  atom 0 sites 12..16 orbitals 2p
```

- We have a weight file with labels “all Fe” “all O” and “K s” created by some other tools. Lets create one weight with them all summed up but with the “all Fe” weight being multiplied with factor 2

```
name all
  labels 'all Fe' fac 2
  labels 'all O' 'K s'
```

- We have a full-relativistic calculation and created pseudo non-relativistic Y_{lm} projections. Weights of Fe 3d are obtained via

```
name Fe_up
  atom Fe sites whatever sites orbitals 3d spin up
name Fe_dn
  atom Fe sites whatever sites orbitals 3d spin dn
name Fe
  atom Fe sites whatever sites orbitals 3d spin both
```

- We want all orbitals of Fe site 2. (This would add all orbitals whose labels start with “Fe(002)” including all spins for pseudo-nonrelativistic weights. If you want the spins separately add the orbitals explicitly)

```
# all weights whose label reads Fe(002)...
name 'All Fe'
  atom Fe sites 2 orbitals all
# all essential spin dn Fe orbitals from pseudo-nonrelativistic projections
name 'all Fe dn'
  atom Fe sites 2 orbitals 3s 4s 3d 4d 4p spin dn
```

The way the program works is to assemble all labels “El(site)orb[spin]” which can be created from the definition and sum them with optional factors. The resulting labels must exist in the file header. If the “labels” keyword is used the file header must contain these labels.

Quotes (“ or ’) can be used to include spaces into names and labels, but they are not needed when no space is contained in a name or label. Comments start with # and end at the line end. Commas can be used to separate list elements but are not needed. An = sign can be added after keywords for convenience (name=Fe instead of name Fe). The range specifier 2..6 is useful and expands to 2,3, ... 6 for long integer lists.

There is the option (ewindow) to define an energy window, which when defined restricts the band written to the output file to bands, which are not completely outside this window. This saves time and makes the resulting files smaller.

The program writes the labels it selected to stdout. Please check the list.

Formal grammar:

- #comments
can appear everywhere
- weightinfile file_name_of_input_weight_file
this will define, which weight file (e.g. file+bweights...) is used as input. If the commandline option -f is used the command line argument file name will be used instead.
- weightoutfile file_name_of_resulting_weight_file
this will define, which weight file (e.g. file+bwsum...) is used as output. If the commandline option -o is used the command line argument file name will be used instead.

- `ewindow emin emax`
remove all bands, which are completely outside this energy window (in eV). This makes the resulting file smaller.
- `name somename`
starts a new output weight adding all the weights defined by `labels-` or `atom-` keywords after this until the next `name`-keyword or end of file is met.
- `labels list_of_input_weight_files_labels [fac number]`
extract and sum all weights corresponding to the labels given in the list. These labels must appear in the input weight file header. Optionally a factor can be specified, which is multiplied to each resulting input weight before adding them all up.
- `atom element sites list_of_int_or_ranges orbitals list_of_orbitalnames [spin up|dn|down|both] [fac number]`
select all input weights with labels `El(site)orb[spin]` multiply them with the optional factor and add them to the output weight defined in the previous `name` clause. The site list can be a list of integers or ranges (e.g. `4..8`). Orbital names can be
 - `nl:` selects all orbitals nlm . Example `3d`: results in `3d-2`, `3d-1`, `3d+0`, `3d+1` and `3d+2`.
 - `nlm:` select orbital nlm . Example `3d+0`: results in `3d+0` (the plus sign matters!)
 - `nlj:` select all orbitals $nlj\mu$. Example `3d3/2`: results in `3d3/2-3/2`, `3d3/2-1/2`, `3d3/2+1/2` and `3d3/2+3/2`
 - `nljm:` select orbital $nlj\mu$. Example `3d5/2+3/2`: results in `3d5/2+3/2`
 - `all:` select all labels, which are specified by the `atom` and `sites` keywords resulting in labels `El(site)...`

Note, that the orbital name `nl` will select the non-relativistic names. If you want to add the whole 3d-shell in a full-relativistic case use `"3d3/2 3d5/2"` to get the whole `nl`-shell.



Chapter 9

Graphical interface: XFPLO

9.1 Structure viewer

Structures can be loaded via

```
XFPLO =.in
XFPLO =.xstr
```

Fresh start in structure mode

```
XFPLO -str
```

Structures can be manipulated in the symmetry dialog. From there one can export `=.in`, which also involves a symmetry update of `=.in`. The whole picture (not just the structure) can be saved into `=.xstr`.

9.2 Fermi surface viewer

A fresh start in fermi surface mode is

```
XFPLO -fs
```

One can save all settings (but not the +band data ... to big!) in `=.xef`, which can be loaded via

```
XFPLO =.xef
```

One can also define the path through the Brillouin zone for band structure plots in the Fermi surface mode. To achieve this first you need at least a `=.in` (use FEDIT or the symmetry dialog in the structure viewer to create this.) Then open

```
XFPLO -fs
```

Now, switch off the Fermi surface display via the Fermi-surface button and switch on the high symmetry points (Fig. 9.1). Open a dialog: Menu plot → high-symmetry-points. Switch on user-defined. Load default if you want. Select a point in the list. Click on the pick button, select current, click on a green point in the Brillouin zone, hit enter or press accept. You must have changed a point. Use pick-button → to create a new point. Use F2 in the list to edit or just start typing. Export to or import from `=.in`, if needed.

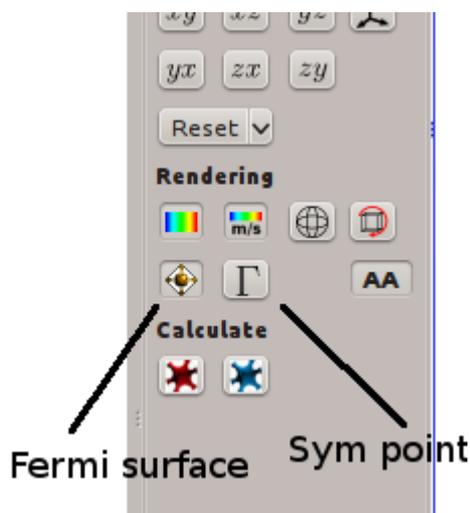


Figure 9.1: Fermi-surface and sym-points buttons.

9.3 Molecular/individual band weights

The default method to create an orbital projected bandstructure (fatbands/band weights) is to switch on band weights in FEDIT. The more flexible way is to use the band weight editor mode of XFPLO. On a fresh start use `XFPLO -bw` or with an existing file `XFPLO =.bwdef` to edit the desired band weights. There are help screens in the program.

One can use this interface to request single weights for selected atoms, which would help to reduce the file size of the usually big file `+bweights`.

One can also create molecular orbital projectors, which are linear combinations of atomic orbitals with certain coefficients, which determine the bonding character and symmetry of the molecular orbital.

9.4 Density mapper

This is not well tested and was implemented on special request of some user.

An existing calculation (call it A) with a `=.dens` file can be a good starting point for a modified structure (call it B). However, since the structure of B determines the dens-file of B the dens-file of A cannot be used in B. You can use XFPLO to define the file `=.densmap`.

We assume that calculation A exists and that the structure of calculation B is already set up (`=.in` exists).

1. Go into the directory of the the new calculation B.
2. Call `"XFPLO =.in"` to display the structure.
3. Open tools→density-maper.
4. Click button "Open old structure" and select the `=.in` of directory A. This will open a new structure view, this time of A.
5. In the mapper table click on an atom of new structure B. The atom will be highlighted. Now click on the atom fo structure A, whose density shall be copied onto the selected B-atom in the dens-file to be created.

6. You can use the “Copy this atom” button to copy this definition to all B-atoms with the same element. Proceed with all B-atoms, which have an equivalent in the A-structure. You can leave some B-atoms un-mapped, in which case a default starting density for these missing atoms will be produced by FPLO. You can also open other old structures and map atoms from there.
7. Save the file in the directory of calculation B (name `=.densmap`).
8. Quit XFPLO.
9. Run FPLO in the directory of B until it stops after creating a new `=.dens`.
10. Zip or rename or delete `=.densmap`.
11. You can now start the B calculation with the freshly mapped dens-file.

Note, that only the spherical part of the local site densities are copied during mapping. Hence, the created mapped density is not necessarily a good starting point.

9.5 Help

Inside XFPLO there are help screens, which explain basic operations in most places which are listed here: [../Xfplo/xfplo.pdf](#).



Chapter 10

Plotting program: XFBP

Use this to plot band structures/DOS and stuff like this. There are help screens (main window and Script editor (Transform Dialog), which explain basic operations:

The new version of the documentation for XFBP is in an external file [../Xfbp/xfbp.pdf](#)



Chapter 11

Occupation matrix manipulator: DMATEDIT

LSDA+U uses onsite occupation number matrices for certain nl -shells (defined by the user in FEDIT). These matrices are saved in `=.dmat_init` and serve as input for consecutive runs. Sometimes the user needs to see/edit them in some local axes and/or complex harmonics basis in order to prepare starting conditions for new runs in order to force the calculation into one of the many possible local (meta) stable solutions of LSDA+U. This is provided by this program. It needs the file `=.dmat_init`, which gets created once LSDA+U was switched on and FPLO was running at least once after switch on. It reads this file, lets you manipulate it and saves it back on Save/Quit. **Note, that the program always saves the file `=.dmat_init` in real harmonics basis of in the global cartesian coordinate system. It uses transformations defined by all the settings in the dialog to display the matrices appropriately in local axis and chosen harmonics, but does save the matrices after transforming them back into the global system.** That means the file can look different from what's shown on screen. In full-relativistic calculations the spin density matrix $n_{mn}^{s/s}$ can be non-diagonal. It is defined within global real harmonics but in the spin frame corresponding to the xc-field quantization axis specified in the FEDIT main menu. This means that its off-diagonal part $n_{mn}^{\uparrow\downarrow}$ should be small. If this is not the case, especially if the diagonal of $n_{nn}^{\uparrow\downarrow}$ is sizeable, the colinear approximation of the full-relativistic mode is violated by LSDA+U. The off-diagonal part (denoted "Spin mixed") is only shown if present and only the up-down block. Note, that the occupation numbers define the shell occupation, shell spin moment and shell L_z -moment.

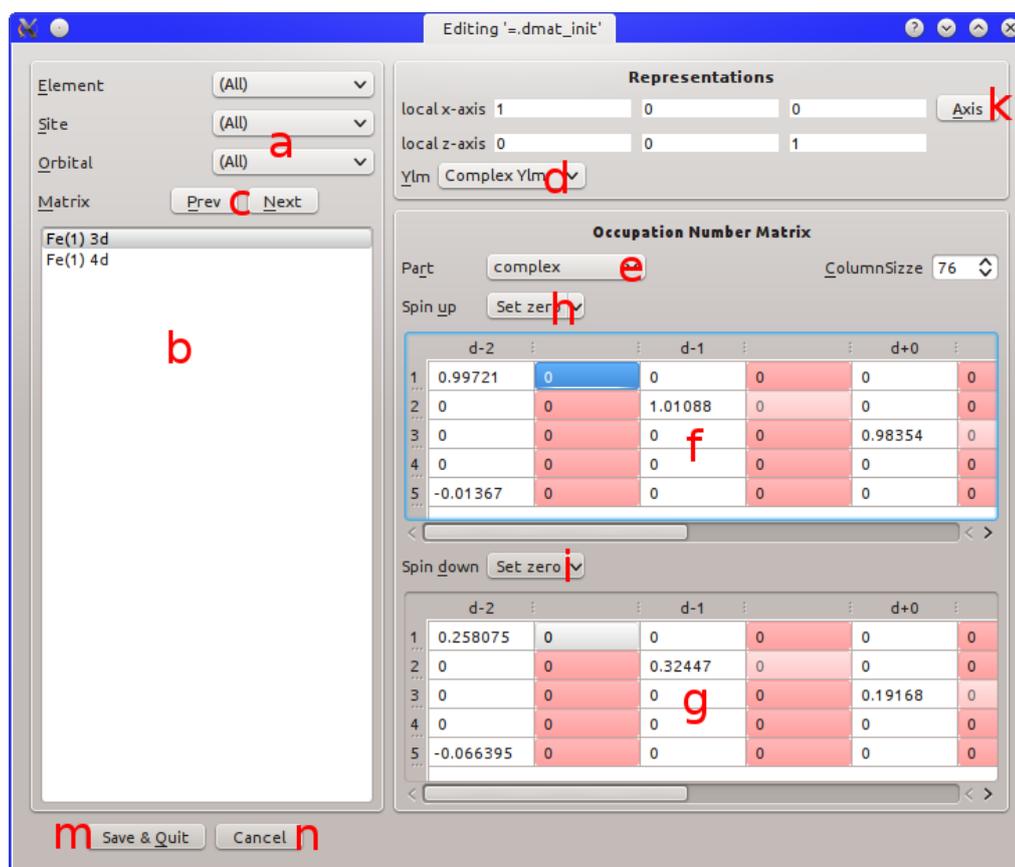


Figure 11.1: DMATEDIT for non-full-relativistic case

- a) Filters defining, which matrices are shown in the occupation matrix list (b) and can be browsed through via the Prev and Next buttons (c). Only the shells for which LSDA+U was actually enabled are shown in this list.
- b) Occupation matrix list filtered by (a)
- c) these buttons browse through the matrices shown in the list (b).
- d) Switch between real and complex spherical harmonics as basis for the matrices.
- e) Chose to show the real, imaginary or both parts in the matrices (f,g)
- f,g) Spin up and down occupation number matrix for the selected shell. Here you can edit the matrices!
- h,i) Set default values. This is a tool button. The arrow to the right allows to select which default it sets. Clicking on the button part sets the currently selected default
- k) Change the local axis in which the harmonics are defined. This is needed for instance if the local environment of the atom is rotated versus the global cartesian system.
- m) The matrices are saved back to `=dmat_init` and the program exits.
- n) (Hotkey `<Escape>`) quit the program without saving.

DMATEDIT saves local axes and such information in `dmatedit.ini` in the current directory.



Chapter 12

Optics: FOPTICS

Optical functions are basically variations of the energy dependent $\mathbf{q} = \mathbf{0}$ dielectric tensor $\varepsilon_{ij}(\omega)$. It consists of an intra- and an inter-band part. The intra-band part is usually taken from the Drude model, since scattering processes, which lead to a finite life-time Γ of the electrons are not included in standard bandstructure calculations.

The Drude model contains the plasma energy tensor ω_{ij}^P , which can be diagonalized resulting in principal axes along which the plasmon has energy $\omega_{1,2,3}^P$ and a life-time $\Gamma_{1,2,3}$. The life-times are user input parameter and the plasmon energies can be chosen by the user as well. However, the plasmon energies and principal axes also get calculated by FPLO and are written to the output and into the file `+plasmon`. Note, that in spin-polarized calculations the principal axes can actually differ between different spin directions (spin up, spin down and total spin sum), which is why all three cases have their own plasmon information in `+plasmon`.

The inter-band part is more expensive to calculate and depends on the actual band structure and on optical matrix elements. The implementation actually only calculates $\text{Im}\varepsilon(\omega)$ for $\omega \geq 0$, since $\text{Re}\varepsilon(\omega)$ can be obtained via the Kramer-Kronig relation. FPLO calculates the optical matrix elements \mathbf{p} and performs the k -integration in the expression

$$\text{Im}\varepsilon_{ij}^{\text{inter}}(\omega) = -\frac{1}{\omega^2} \frac{4\pi^2}{V} \sum_{\mathbf{k}, \bar{n}, n} [f(\varepsilon_{\mathbf{k}, n}) - f(\varepsilon_{\mathbf{k}, \bar{n}})] \delta(\varepsilon_{\mathbf{k}, n} - \varepsilon_{\mathbf{k}, \bar{n}} - \omega) \mathbf{p}_{i\mathbf{k}n\bar{n}} \circ \mathbf{p}_{j\mathbf{k}\bar{n}n}^*$$

and writes the result into the file `+imeps`. This file is not meant to be used directly, but one can look at it via XFBP for convergence control. You can control this process via the OPTICS submenu in FEDIT (read the help screen).

Caveats: The inter-band k -integral is not very smoothly converging especially if many small Fermi surfaces are present. Always check k -point convergence of `+imeps`. Furthermore, FPLO is a small basis method, which means that there are not very many unoccupied bands, which means that optics gets less and less accurate the higher the energy ω is.

After FPLO has created the file `+imeps` some optical functions can be created in a separate step, in which also the intra-band part gets added. The program FOPTICS is a command line tool. It has command line options, which are available via `FOPTICS -h`:

usage:

```
foptics22.00-62-x86_64 [-i fploutput_imag_part_interband_eps] ...
    [-os suffix_for_output_files] [-h] ...
    [-gamma gamma1 [gamma2 ... gamman]] ...
    [-omega omega1 [omega2 ... omegan]] ...
    [-skip nskip] [-nointer] [-nointra] [-ni]
```

fplooutput_imag_part_interband_eps: normally +imeps
 omega: the plasmon frequencies for the intraband contributions in eV
 gamma: the broadening (life time) for the intraband contributions in eV
 nskip: take only every nskip data point
 -nointer: only take the intraband contributions
 -nointra: only take the interband contributions
 -ni = not interactively: take values from +plasmon and from -gamma and -omega
 but do not ask for not explicitly specified values.

There are 3 gamma values: for each principal axis one.
 There are 3 omega values along the principal axes for each spin direction
 and for the total (spin sum). This is so, since the principal axes
 need no coincide for spin up, spin down and the spin sum.
 In non-spin-polarized cases only spin1 is used and equals the spin sum.

This program reads inter-band output from FPL0 (+imeps)
 and calculates some optical functions. The user can define
 plasmon frequencies and broadenings to include the intra-band
 contributions.

The program also reads the file +plasmon to extract the calculated
 plasmon energies and the principal axes of the plasmon tensor.
 The user can overwrite the plasmon energies by interactive input
 or by the option -omega.

The output functions are

re_eps: real part of epsilon
 im_eps: imaginary part of epsilon
 re_sigma: real part of the optical conductivity in inverse Ohm*cm
 loss: the loss function

For the individual spin contributions _spin1/_spin2 is appended to the
 file names. (Only for spin polarized cases.)

If an output suffix (-os) is defined it will be appended to all file names.

One can also just call the program without options. It reads +imeps to obtain $\text{Im}\varepsilon^{\text{inter}}(\omega)$ and +plasmon
 to obtain the eigen decomposition of $\underline{\omega}^P = \sum_i \mathbf{Z}_i \omega_i^P \mathbf{Z}_i^T$, where \mathbf{Z}_i is a principal axis of the tensor. The
 program writes the plasmon and life-time information to standard out and asks the user for input of ω_i^P and
 Γ_i unless option -nointra or -ni was set or values were provided on the command line. The command line
 option -gamma reads up to 3 Γ_i , one for each principle axis. These Γ_i are used for all spin directions. The
 command line option -omega reads up to 9 ω_i^P , the first three are $\omega_{1,2,3}^{P\uparrow}$, the next three are $\omega_i^{P\downarrow}$ and the last
 three are $\omega_i^{P(\uparrow+\downarrow)}$. If not all $\omega_i^P(\Gamma)$ values are provided the program will interactively ask for user input for
 the non-specified values. The default ω_i^P values are taken from +plasmon. If option -ni (not interactively)
 is set, all values specified via -omega and -gamma are used and the not explicitly specified values are taken
 from +plasmon. So, interactivity can be avoided by either specifying all values via the options (3 Γ and 3 ω
 (non-polarized) or 9 ω spin-polarized) or by using -ni, which is usefull for falling back to the default values
 from +plasmon without having to type enter for all questions asked. Play with it to get the idea.

FOPTICS will use the Kramers-Kronig transformation to calculate $\text{Re}\underline{\varepsilon}^{\text{inter}}$ from $\text{Im}\underline{\varepsilon}^{\text{inter}}$. Then the intra-band part Eq. (12.1) is determined for which the plasmon tensor is constructed from the principal axis \mathbf{Z}_i and the (user-specified) parameters Γ_i and ω_i^P . Finally,

$$\underline{\varepsilon} = \underline{\varepsilon}^{\text{intra}} + \underline{\varepsilon}^{\text{inter}}$$

is calculated and the optical functions are derived from it and written into separate files. Note, that the two parts can be switched off via the options `-nointer` and `-nointra`. Currently, there are the following optical functions (see Sec. 12.1)

function	file name
$\text{Re}\varepsilon_{ij}(\omega)$	re_eps[_spin1 _spin2]
$\text{Im}\varepsilon_{ij}(\omega)$	im_eps[_spin1 _spin2]
$\text{Re}\sigma_{ij}(\omega)$	re_sigma[_spin1 _spin2]
$L_{ij}(\omega)$	loss[_spin1 _spin2]

Table 12.1: Optical functions

These files contain comments for user orientation and they can most conveniently be displayed with XFBP. Note, that only the non-zero tensor elements are written. A symmetry analysis of the inter-band part is used to determine these tensor elements. In this it is assumed that the intra-band part must have the same symmetry. The options `-i` and `-os` are useful if several different settings (e.g. k -points) are used and the file `+imeps` was renamed into e.g. `imeps_12_12_6` and `imeps_42_42_12` for the different runs by the user. Then FOPTICS `-i imeps_12_12_6 -os 12_12_6` reads `imeps_12_12_6` and produces `re_eps_spin1_12_12_6` and so on.

Example A1 We make a standard calculation for Al as explained in getting started. Then we go to the OPTICS submenu of FEDIT and switch on optics and set the upper energy bound to 20 eV (this is for demonstration of the plasmon peaks, the interband part is not really accurate at such high energies!). We re-run FPLO. The UNIX command

```
ls -ltr
```

shows that the file `+imeps` got created. For orientation open it via

```
XFBP +imeps
```

Copy it:

```
cp ./+imeps ./+imeps_12
```

(to indicate the default $12 \times 12 \times 12$ k -mesh). Change the k -mesh in FEDIT to $42 \times 42 \times 42$. Re-run FPLO. (This will require a few steps to achieve the self-consistency for this k -mesh.) Copy it:

```
cp ./+imeps ./+imeps_42.
```

Load both files

```
XFBP +imeps_12 +imeps_42
```

You see the convergence issue. Make another calculation for a $60 \times 60 \times 60$ k -mesh, copy the file and compare the +imeps files. (Aluminum really has tiny Fermi surfaces.)

Now, run

```
FOPTICS -i +imeps_60 -os 60
```

and hit enter for all questions or use option -ni to take the default values. Open the resulting loss function

```
XFBP loss_60
```

(the suffix _60 came from the -os 60 option). Compare the position of the plasmon peak at 15eV with the bare plasmon frequency 12.3eV in the FOPTICS output. This shift is due to the inter-band part. Now, calculate the intra-band only loss function:

```
FOPTICS -i +imeps_60 -os 60_intra -nointer
```

Now, the plasmon peak is at the bare energy 12.3eV. At last calculate the inter-band only loss function:

```
FOPTICS -i +imeps_60 -os 60_inter -nointra
```

(Beware of the correct placement of inter and intra!!) Compare the three:

```
XFBP loss_60 loss_60_intra loss_60_inter
```

You must see something like Figure 12.1.

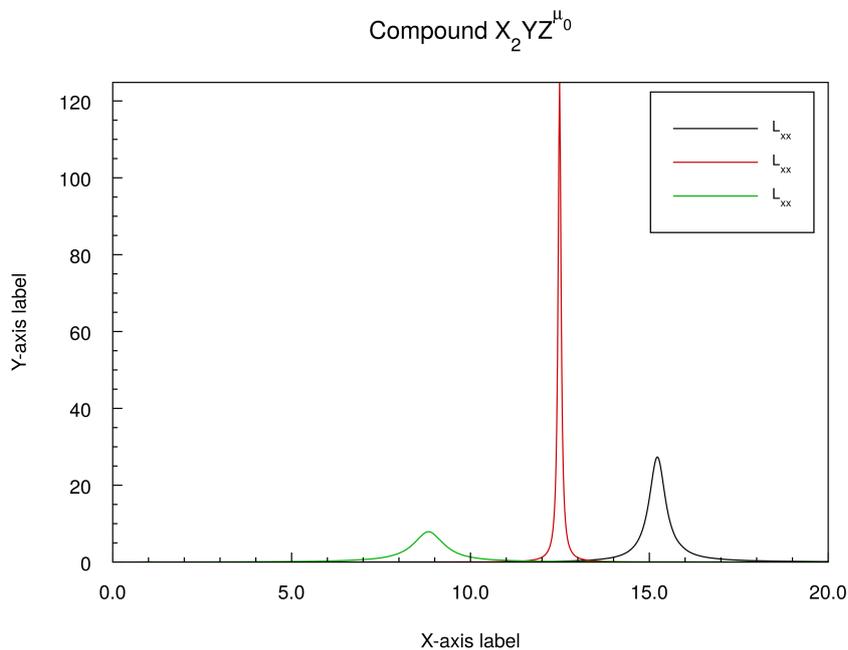


Figure 12.1: Total, intra-band and inter-band loss function for Al.

This example is also discussed by the Wien2k implementation. Finally, you can have a look at all the other functions (Note that the intra-band part of ϵ diverges at $\omega = 0$).

12.1 Optical functions

12.1.1 General

The immediate function is the optical dielectric function (without momentum transfer)

$$\boxed{\varepsilon(\omega) = \varepsilon^{\text{intra-band}}(\omega) + \varepsilon^{\text{inter-band}}(\omega)}$$

where the intra-band expression must be approximated via a Drude model and the interband part is the expensive thing the optics-module is all about. ε is connected to the optical conductivity via

$$\sigma(\omega) = -i \frac{1}{4\pi} \omega (\varepsilon(\omega) - \mathbf{1})$$

which makes σ a measure of the imaginary part of ε .

$$\boxed{\text{Re}\sigma(\omega) = \frac{1}{4\pi} \omega \text{Im}\varepsilon(\omega)}$$

$$\text{Im}\sigma(\omega) = \frac{1}{4\pi} \omega (\mathbf{1} - \text{Re}\varepsilon(\omega))$$

The Loss function is defined as

$$\boxed{L_{\alpha\beta} = -\text{Im} \left(\frac{1}{\varepsilon(\omega)} \right)_{\alpha\beta}}$$

12.1.2 Intra-band (Drude)

The intraband contribution is approximated by

$$\boxed{\underline{\varepsilon}^{\text{intra}}(\omega) = \mathbf{1} - \frac{\omega_P^2}{\omega(\omega + i\Gamma)}} \quad (12.1)$$

which gives

$$\boxed{\text{Re}\underline{\varepsilon}^{\text{intra}}(\omega) = \mathbf{1} - \frac{\omega_P^2}{\omega^2 + \Gamma^2}}$$

with a root at

$$\omega_0 = \sqrt{\omega_P^2 - \Gamma^2}$$

and

$$\text{Im}\underline{\varepsilon}^{\text{intra}}(\omega) = \frac{\omega_P^2 \Gamma}{\omega(\omega^2 + \Gamma^2)}$$

which results in

$$\boxed{\text{Re}\sigma^{\text{intra}}(\omega) = \frac{1}{4\pi} \frac{\omega_P^2 \Gamma}{\omega^2 + \Gamma^2}}$$

$$\text{Im}\sigma^{\text{intra}}(\omega) = \frac{\omega}{\Gamma} \text{Re}\sigma^{\text{intra}}(\omega)$$

For the loss function we get (in the diagonal principal axis frame of ω^P)

$$L_{\alpha\alpha} = \omega \Gamma \frac{\omega_{P\alpha}^2}{(\omega^2 - \omega_{P\alpha}^2)^2 + \omega^2 \Gamma^2} \quad (12.2)$$

with the maximum at

$$\omega_{\text{peak}} = \frac{1}{\sqrt{6}}\omega_{P\alpha} \sqrt{2 - \frac{\Gamma^2}{\omega_{P\alpha}^2} + \sqrt{16 - 4\frac{\Gamma^2}{\omega_{P\alpha}^2} + \frac{\Gamma^4}{\omega_{P\alpha}^4}}}$$

$$\omega_{\text{peak}} \approx \omega_{P\alpha} \left(1 - \frac{1}{8} \frac{\Gamma^2}{\omega_{P\alpha}^2}\right)$$

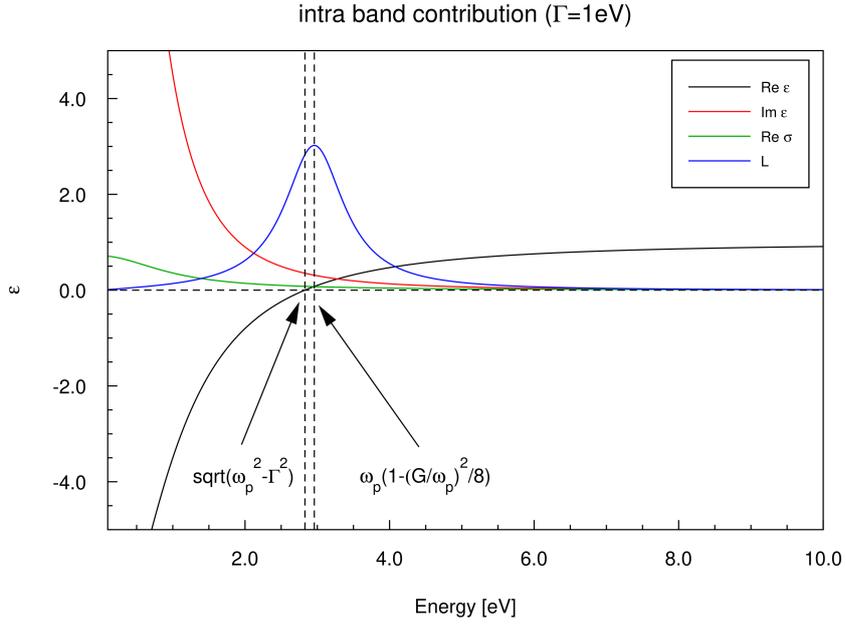


Figure 12.2: Intra band contribution for an unaturally large $\Gamma = 1\text{eV}$ (to resolve the small shifts of ω_P .)

12.1.3 Intra-band and constant interband ε_∞

Now, we include a constant approximation for the interband contribution ε_∞ .

$$\begin{aligned} \underline{\varepsilon}(\omega) &= \mathbf{1} - \frac{\omega_P^2}{\omega(\omega + i\Gamma)} + \varepsilon_\infty \\ &= (1 + \varepsilon_\infty) \left(\mathbf{1} - \frac{\tilde{\omega}_P^2}{\omega(\omega + i\Gamma)} \right) \end{aligned}$$

where we defined the renormalized plasmon frequency as

$$\tilde{\omega}_P = \frac{\omega_P}{\sqrt{1 + \varepsilon_\infty}}$$

This means that in all expression we have to replace $\omega_{P\alpha} \rightarrow \tilde{\omega}_{P\alpha}$ and devide (multiply) by $\frac{1}{1+\varepsilon_\infty}$. Eq. (12.2) then gives

$$L_{\alpha\alpha} = \frac{1}{1 + \varepsilon_\infty} \omega \Gamma \frac{\tilde{\omega}_{P\alpha}^2}{(\omega^2 - \tilde{\omega}_{P\alpha}^2)^2 + \omega^2 \Gamma^2}$$

with the peak at

$$\omega_{\text{peak}}^{\text{intra}+\varepsilon_\infty} = \tilde{\omega}_{P\alpha} \left(1 - \frac{1}{8} \frac{\Gamma^2}{\tilde{\omega}_{P\alpha}^2} \right)$$

which is better suited to determine the plasmon peak than extracting the divergent Drude part from σ .



Chapter 13

The dHvA-Module: FPLO/FDHVA

Note: it is not possible to explain everything in a linear order. It is advisable to read the document two times. In order to calculate dHvA spectra a two-stage procedure has to be followed. In the first stage a high resolution energy iso-surface (IS) is calculated. It serves as input for the second stage, which determines the extremal orbits and produces the spectrum (extremal area vs. field direction). We call the Fermi surface iso surface in the following to stress that it does not need to be at the Fermi level.

13.1 First stage (iso surface)

The iso surface is not calculated via the usual FPLO interfaces because the accuracy demands for dHvA are higher. Instead we use a different module, which calculates the iso surface via mesh refinement.

Initially the primitive unit cell is transformed into an equivalent cell, which is as isotropic as possible. This defines three cell basis vectors (or axis). A coarse initial mesh of $n_1 \times n_2 \times n_3$ micro cells is defined for this unit cell on which the Hamiltonian is diagonalized (band energies are determined). In fact the micro cells themselves are subdivided into tetrahedra in order to use linear interpolation. Then each micro cell, which contains the iso surface (the tetrahedra edges are cut by the iso surface) is subdivided into 2^3 smaller cells by bisecting each axis direction, which introduces new mesh points (the bisecting points) on which in turn the band energies are determined. Now, again the resulting micro cells which contain the iso surface are determined and another subdivision step is taken until a maximum number of bisections is reached. In many cases this procedure leads to a high resolution iso surface without the necessity to diagonalize the Hamiltonian on each mesh point of the finest subdivision. It is clear from this description of the algorithm that at the end only iso surface parts are resolved, which are “seen” by the initial subdivision. If an iso surface part is contained completely within one of the initial mesh micro cells it will not be visible after a number of refinements either since only iso-surface-cutting micro cells are subdivided. In the moment the crystal symmetry is not used. This has both technical as well as conceptual reasons. The latter being that the module allows band weight dependent energy shifts for the adaptation of the dHvA spectra. In general band weights do not have the full symmetry of the lattice. Hence, the use of symmetry would be a hinderance. If the energy shift option is not used, symmetry would improve the performance, but only for highly symmetric systems.

The process of bisecting the mesh can be done iteratively by the user. For this purpose all energy/band-weight data are stored in a cache file (`+isoergcache...`) during the iso surface stage. If the resulting iso surface is not accurate enough a higher bisection level can be set and in a re-run the data for all previously calculated mesh points are read from the cache and not recalculated. Essentially, only the points needed for the now required finer subdivision are calculated. This caching of course only makes sense if the self consistent calculation (density) did not change in between. Also all band weight options for the iso surface stage should not change. The code is able to detect certain changes to the input data (energy window, band

weight options) and invalidates the cache automatically. However, if other data change (e.g. the density) the user is responsible to delete the cache file by hand before a re-run. A program termination during a cache write can lead to incomplete data in the cache, which is detected in a re-run. An error message is written and the corresponding mesh points data are recalculated. So, you can safely interrupt an IS stage run. In case that a Wannier function model is used to extract the iso surface the cache file is named `+isoergcache_wan`.

If the user wants to change the iso level the cache file can be re-used. For small changes the existing mesh points are likely to already surround the slightly changed iso surface. This leads to less necessary diagonalizations as compared to a fresh run without a cache file.

The initial subdivision must be chosen such that it is as homogenous as possible. Note, that the iso-cell is not the primitive unit cell but a modified one. This cell together with a suggestion for a proper subdivision is written to the output. All lines referring to the iso surface calculation are prefixed with "ISO:". There usually is an initial output section and the output section for the actual calculation. Always check these parts of output for a proper setup. The relevant part for the initial mesh looks like this:

```
ISO:
ISO: Reciprocal iso cell vectors/(2*pi):
ISO: g1:    -0.09903    -0.17153    0.00000, |g1|:    0.19807
ISO: g2:     0.19807     0.00000    0.00000, |g2|:    0.19807
ISO: g3:    -0.00000     0.00000    0.15025, |g3|:    0.15025
ISO:
ISO: Most homogenous initial subdivision would be: 10 10 8 or multiples of these.
ISO:
```

The default way of performing the iso surface stage is by defining a number of input values and by running FPLO. FPLO will do what it usually does (calculating the potential, the local orbital integrals (LOI) and diagonalizing the Hamiltonian) until after the population analysis, where it jumps into the iso surface module (if switched on). Only after the population analysis the LOI and Fermi energy are known. This means that for each IS stage run it has to go through the diagonalizations of at least one SCF step as for the calculation of the default band structure output (`+band...`). There is an option to circumvent this. For that to take effect you need to define a proper Wannier function (WF) model. Make sure that it reproduces the band structure around the desired iso value (Fermi energy) as good as possible (or the dHvA spectrum will not be exactly the same as from an FPLO run). The IS stage input can be told to use the output of the WF modul in which case FPLO will jump into the IS module early in its run. (The SCF step is avoided).

The result of the IS stage run are files which contain the iso surface. They have names `+iso_b..._p..._spin...` where the ellipses stand for numbers. The numbers after `_b` denote the band index of the corresponding band, `_p` denotes the number of the part of the sheet (if the sheet for this band index has disconnected parts there can be more than one part.) and `_spin` denotes the spin. For full-relativistic calculations the spin suffix is missing, since spin is not a good quantum number in this case. In the default setup the program will automatically determine the bands, which cross the iso level (usually the Fermi energy). Optionally, the user can request a set of specific bands. `+iso_b...` files will only be created if the band has an iso surface for the given iso level. To spell it out explicitly: If the iso surface of a certain band number has disconnected parts, they are treated as separate entities and get their own file.

After the IS stage run is performed individual iso surface parts can be visualized by loading them into XFPLO

```
xfplo +iso_... +iso_...
```

More than one file can be given as argument. Or one can load all

```
xfplo +iso_*
```

Additionally, the option `-raw` can be given

```
xfplo -raw +iso_... +iso_...
```

in which case the iso surface data are shown as they are stored in the file, that is only in the iso cell which was setup at the beginning. Without this option the data are periodically extended to fill the boundary of the main display cell (usually the black frame). The loading with the `-raw` option is faster since no clipping takes place. There is one peculiarity in the raw mode : if an iso surface part is closed, .i.e. completely contained within one (possibly shifted) unit cell the file data will contain the iso surface part in one piece, which means its unit cell will be shifted accordingly if needed. If the iso surface part is open it will be contained in the iso unit cell which was set up in the beginning.

When loading iso files the way shown above the default xfplo Fermi surface is switched off. It can be switched with the button shown in Fig. 13.1. The default coloring shows the magnitude of the Fermi velocity (at the chosen iso level [the one in the dHvA FEDIT submenu not he one in XFPLO]). If the band weight options where



Figure 13.1: Left: Fermi surface button in the tool box of XFPLO. Right: extern color values checkbox and state spinbox in the plot->colors menu in XFPLO.

set in the dHvA FEDIT submenu the coloring can be changed to depict as chosen band weight. For this go into the `plot -> colors` menu in XFPLO and switch on `extern color values`. The `state` spinbox allows to select the number of the band weight to be shown. The name of the weigh should appear in the legend (unless switched off). If the `state` number is invalid it is announced in the legend as well. It is in principle possible to load `+iso_...` files together with the default Fermi surface. In this case the legend can only show one of the two things. Consider it to be undefined behaviour. Currently the `+iso_...` files cannot be saved in `=.xef`, they can only be loaded via the command line arguments as shown above. One can however, define other settings in XFPLO (e.g. the `extern color` settings) and save them from XFPLO (usually in `=.xef`). In this case load the files via

```
xfplo =.xef +iso_...
```

in order to restore the settings previously saved in `=.xef`. The default coloring with the Fermi velocity helps to judge the quality of the iso surface. The smoother it is the better will the subsequent dHvA run perform (see Sec. 13.2). Of course there are limits due to calculation time. The Fermi velocity (or gradient) is used in FDHVA to determine the connectivity of the individual extremal areas. Consequently, a good gradient is vital for a good dHvA spectrum.

The `+iso_...` files will be deleted at the beginning of each IS stage run to avoid confusions between consecutive runs to optimize the refinement. This behaviour can be disabled by setting a flag in the dHvA FEDIT submenu. The background is that the number of parts for each iso surface sheet depends on the mesh size. E.g. iso surfaces with pointy edges or tips can end up producing the main iso surface and tiny disjointed parts at the tips or edges. If you cannot see these parts in XFPLO have a look at the file size of the `+iso_...` files. If they are small compared to the files sizes of other parts it is an indication that the corresponding part is also small. There is an option to drag band weight information along with the band energies. This can be helpful for visualization or for manipulating the band energies via a band character dependent energy shift. FPLO has several ways of producing band weights: the old style where a transformation of the global quantization axes

can be defined in the bandplot FEDIT submenu or the more flexible approach the molecular/individual fat bands Sec. ??, ?? and 9.3. The later approach allows individual quantization axes for each weight and it is necessary in the context of the IS stage if an *nlms* basis is wanted for full relativistic calculations (where the default basis is spherical spinors $[nj\mu]$). FPLO will transmit the band weights defined either via the old style (bandplot FEDIT submenu) or via `=.bwdef` (also there) to the IS module. Let's call them raw weights.

If the Wannier function model option was chosen, the weights for all WFs will be transmitted to the IS module. The `=.bwdef` file options is not available in this case, even if such a file is defined in FEDIT. Local axes transformations (if wanted) must be defined when defining the Wannier functions themselves.

Note, that the raw weight setup takes place in the bandplot submenu not in the dHvA submenu. (Similarly, the energy window options from the bandplot menu will be used in the IS module, but only if no WF model is used.) The raw weights will be written into the cache file.

Additionally, one can define another set of band weights, which is obtained by adding some of the raw weights. This is achieved by editing an `=.addwei` file and entering it into the appropriate field in the IS FEDIT submenu (see below). Also see section 77 for information about this file. If such an adding step is defined by the user we obtain another set of weights (added/final weights). These weights are written into the `+iso_b...` output files. Hence, when the energy window information or the `=.bwdef` file options are changed in FEDIT the cache file will lose its validity, while altering the weight-adding option will not.

Of course the weights defined in the `=.addwei` file (which one can name as wished) must be build out of the set of raw weights. The information about the available raw weights is written to the output in the "ISO:" section at the beginning of the IS stage calculation (after the population analysis). Only the weight definitions in this file are used by the IS stage. The energy window and file information in `=.addwei` are ignored for this purpose.

If the adding step is omitted, by not entering a file name into the "addwei file" field in the dHvA submenu, the raw weights become the final weights.

Figure 13.2 summarizes the flow chart of the weights.

Finally, one can use the final weights to define \mathbf{k} -dependent energy shifts $\Delta\varepsilon_n(\mathbf{k})$, which are defined as a weighted sum of the final/added band weights. This results is a band character dependent energy shift. For the definition of the terms of the energy shift formula one needs the number of the final weight, which enters the term. Final weights are counted in the order as written to the output starting with 1. (See below)

For completeness the density of states (DOS) of each iso surface part is written to the output together with an estimate of the change of the particle number dN due to the energy shift. Example:

```
ISO: =====
ISO: spin summed, band 00011, part 001, dos=          0.07863 [states/eV] dN~=          0.00000
ISO: spin summed, band 00012, part 001, dos=          0.11918 [states/eV] dN~=          0.00000
ISO: spin summed, band 00012, part 002, dos=          0.11918 [states/eV] dN~=          0.00000
ISO: =====
ISO:
ISO: =====
ISO:           Finished Iso-Surface Calculation
ISO: =====
```

The change of the particle number is determined via

$$dN = \int d^3k \delta(\omega - \varepsilon_n(\mathbf{k})) \Delta\varepsilon_n(\mathbf{k})$$

which is the same as: DOS times iso surface average of the energy shift $\Delta\varepsilon_n(\mathbf{k})$. Note, that an IS sheet/part can completely vanish due to a shift, in which case all the electrons/holes of this sheet become unoccupied. The code cannot estimate a dN for such a situation.

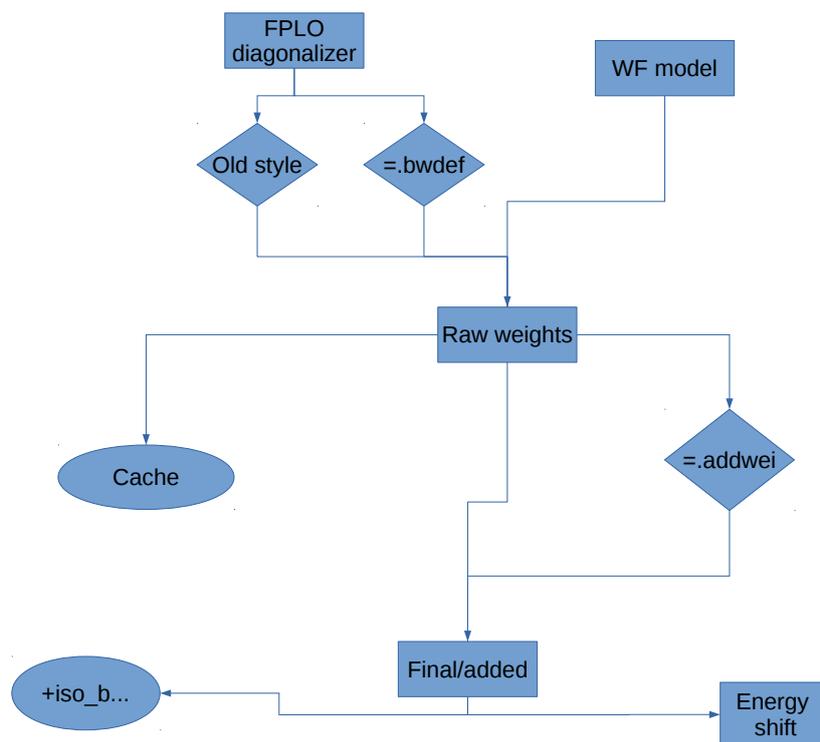


Figure 13.2: Flow chart of weights.

13.1.1 Input options

The input which controls the IS stage is edited via the first part of the dHvA submenu of FEDIT shown in Fig. 13.3.

Iso surface run If this switch is on FPLO will activate the IS stage after the first population analysis. If the Wannier function route is taken the IS stage takes place before the first SCF cycle.

Iso value This is the iso value (Fermi energy) at which to calculate the iso surface. An iso value zero corresponds to the Fermi level of the SCF cycle. The units are eV.

Initial subdivision This is a three values list of the number of intervals along each of the iso cell axes, which will determine the initial mesh in the Brillouin zone (see explanation above).

Bisections This is the maximum number of bisecting subdivisions which will be performed to refine the mesh around the iso surface sheets. A value of zero means that only the mesh from the initial subdivision will be used. If the resulting iso surfaces are not fine enough one can come back and increase this value and make a re-run of FPLO. The cache file `+isoergcache...` will be used to avoid unnecessary recalculations unless a change of some input data invalidates the cache.

Spin-up/down band number A list of band numbers. The default value is an empty list, which triggers automatic determination of the bands, which cross the iso value. This option is useful when the individual iso surface parts have very different sizes, in which case one can use different refinement levels (bisection values) for different iso surface parts. **Warning:** In such a case make sure that you switch off the `delete old files` flag or save the `+iso_b...` files somewhere else, while treating specific bands.

```

e (X)it                                     DHVA                                     (/) Search (H)elp
PREPARE ISO SURFACE

Iso Surface (R)un      : [X]

Iso (V)alue           : 0.
Initial (S)ubdivision : 12 12 12
(B)isections         : 3

spin-(U)p   band number :
spin-(D)own band number :

(W)annier Hamilt. file :

de(L)ete old files    : [X]

n(E)ed bandweights   : [X]
(O) addwei file      : =.addweiiso

Energy shift
(P) number of shift terms : 0

-----
No.   weight-index   factor
-----

DHVA

(N)umber of fields    : 2

-----
[...]
```

STATUS: OK (18.00-52:M-CPA)

Figure 13.3: First part of the dHvA sub menu in FEDIT.

Wannier Hamiltonian file Here you can give the name of a Wannier Hamiltonian file. Default: empty string. This file is created in a Wannier function run of the newer FPLO versions and by default is called `+hamdata`. It contains the real space tight binding model derived from the Wannier functions as requested by the user in `=.wandef`. One can however, also write this file by hand to construct a model. If the file name is given FPLO will branch into another mode after the initialization before the start of the first SCF cycle. It will then calculate the iso surface corresponding to the band structure of the model. This can help to speed things up, in case that a reasonable WF model can be created.

Delete old files As explained above the code will delete all existing `+iso_b...` files in the beginning of each IS stage run to avoid confusion with left over files from previous runs. Uncheck this flag if bands shall be treated individually on users request.

Need bandweights If this flag is on the code will extract band weight information additionally to the band energies as explained above. If this flag is toggled between reruns the cache file will be invalidated. Chose your desired settings early to avoid wasting resources. This flag also needs to be set if energy shifts are used.

Addwei file Default: empty string. Give the filename of a FADDWEI file containing appropriate definitions

for adding the band weights obtained from the iso surface run. If set, the `+iso_b...` files will contain the added weights, otherwise they will contain the raw band weights. Changing the addwei file does not affect the cache. It only affects the `+iso_b...` files and the energy shift function. Only the weight definitions of this file are important not the other settings. For more information see Sec. 8)

Energy shift (expert option) One can chose to define a \mathbf{k} -dependend energy shift. This is done by defining a number of terms which consist of a chosen (added) band weight and a numerical factor. The energy shift is defined as

$$\Delta\varepsilon_n(\mathbf{k}) = \sum_i f_i w_{n,\alpha_i}(\mathbf{k})$$

where f_i are the numerical factors and w_{n,α_i} are the band weights of band n , where α_i numbers the available weights. This shift is not a simple shift of band n against $n + 1$ but depends on the band charachter. The available weights are either the raw weights (if no weight adding file is defined) or the added weights. The output contains a list of available weights.

number of shift terms Set the number of terms in the shift function.

shift term i give a weight index α_i (first weight is number 1) and the numerical factor f_i .

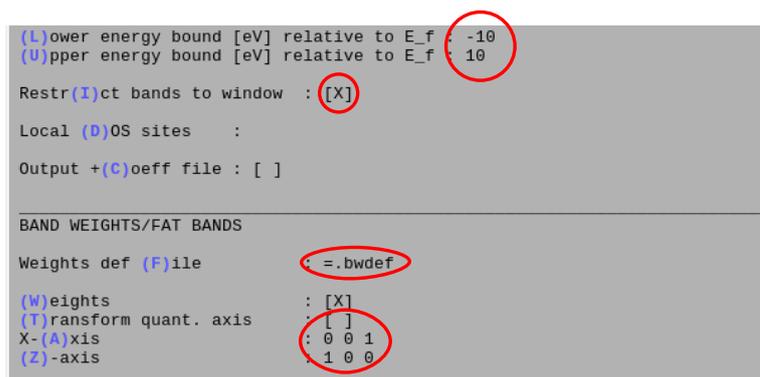


Figure 13.4: Part of the bandplot sub menu in FEDIT.

Additionally, some fields of the bandplot FEDIT submenu are used to control the IS stage (shown in Fig. 13.4) if the Wannier function option is NOT used.

Lower/Upper energy bound If `restrict bands to window` is checked (see item below) this defines the energy window, which is used by the IS stage run to produce smaller `+isoergcache...` files. Changes to the energy window will invalidate the cache file.

Restrict bands to window If checked us the bouds defined in the item above to restrict the data written to the cache file. Changes to this flag will invalidate the cache file.

Weight def file Here one can give a file which contains (default `=.bwdef`) weight definitions. It is usually used to to define compound weights. But it can also be used to define a sub set of weights and to define individual local rotation axes. If this input field contains a file name the resulting raw weights will be taken according to the definitions in this file. This helps reducing the size of the cache file. In full relativistic calculations the default FPLO weights are in $nj\mu$ basis (spherical spinors). If the non-relativisic $nlms$ symmetry is desired the only option is to setup a `=.bwdef` file and enter it here in FEDIT. (See the help screens Sec. ?? and ?? and Sec. 9.3)

`Transform quant. axis` If `=.bwdef` is not used the usual old-style transformation of the global quantization axes is applied for the raw weights delivered from FPLO to the IS stage.

`X/Z-axis` See item above.

13.1.2 Cache file

For the interested party the content of `+isoergcache...` is explained. The file starts with a header

```
HEADER
nband=6
ilower=4
iupper=9
weights="Al(001)2s+0","Al(001)2p-1",...
ENDHEADER
```

There are `nband` bands stored in the file, the first being original band `ilower+1` and the last `iupper+1`. If `weights` is not an empty list there are as many weights stored as there are weight labels in the list.

The file body is a list of data for each mesh point (which ever has been used throughout the refinement). The data of one point are in one single line and are structured like this:

```
spin-index  $k_x$   $k_y$   $k_z$   $\varepsilon_1 \dots \varepsilon_{nband}$   $w_{1,1} \dots w_{1,nweight}$   $w_{2,1} \dots w_{2,nweight}$   $\dots$   $w_{nband,1} \dots w_{nband,nweight}$ 
```

From this header it should be clear under which conditions the cache file gets automatically invalidated.

- Changes to `Lower/Upper energy bound`: this affects `nband`, `ilower` and `iupper`
- Changes to `Restrict bands to window`: this affects `nband`, `ilower` and `iupper`
- Changes to `Need bandweights`: this changes the `weights` label list in the header
- When the list of `weight labels` changes. E.g. if one adds a `Weight def file`, which has a weight label list different from the default list.

One can safely change the following input without deleting the cache file

- `iso value`
- `initial subdivision`
- `bisections`
- `spin-up/down band number`
- `addwei file`
- `energy shift`

The user has to delete the cache file by hand when the following changes

- When information in the `Weight def file` changes, which does not alter the weight label list: e.g. the local axis of the weights. Basically when information is changed, which is NOT reflected by the weight header.
- When `Transform quant. axis` or `X/Z-axis` changes (if no `=.bwdef` is specified)
- When the density changes.

13.1.3 Work flow

Please note that the weight options are not needed for a standard dHvA spectrum calculation. They are needed for visualization purposed (if desired) and for the energy shifts (expert option).

- Converged the SCF calculation
- If you need weights, figure out which ones you will need. If additionally added weights are needed figure that out too.
 - Setup `=.bwdef` (prefered option) and enter the file name into the bandplot submenu of FEDIT.
 - Setup `=.addwei` and enter it into the dHvA submenu for later use.
 - Use standard band structures/fat bands to evaluate the physics of the problem. (Use FADDWEI to test the weight adding in the band structure step.
 - If you use a WF model instead, setup the WFs appropriately and an `=.addwei` if needed. Add the `+hamdata` file name to the dHvA submenu. `=.bwdef` cannot be used in this case.
- Optionally, for orientation you can use XFLO to get a feel for the Fermi surface.
- If weights are needed turn on the corresponding options in the dHvA submenu.
- From a default band structure plot determine the needed energy window. To save disk space and cache load time set the energy window and switch on “Restrict bands to window” in the bandplot submenu of FEDIT.
- Setup the initial subdivision and a bisection of 0 which is fine enough to capture all Fermi surface parts. Make a run and visualize the iso surface.
- Increase the bisections until a desired accuracy is achieved. It might be helpfull to make a dHvA stage run to judge the quality and then to come back and increase the fineness further.
- If energy shifts are used, some iterations are needed to achieve the desired result. In each re-run of the IS stage the chache file should be valid and hence this stage fast (unless the shifts are huge). However you still have to go through one SCF step for each re-run. (Wannier function model might help)

13.2 Second stage (dHvA spectrum)

The second stage of the dHvA module takes the output of the first stage, the files `+iso_b*`, and calculates the dHvA spectrum, i.e. extremal areas versus field directions. Although the input for this stage is also accesible via the FEDIT dHvA submenu, the executable to run this step is different form FPLO it is called FDHVA.

The field directions are determined by a number of main directions \mathbf{B}_i in space. These play a similar role as the high symmetry points in a bandstructure. In between two consecutive main directions a number of intermediate fields is determined by subdividing the angle between the two main directions into a number of smaller intervals. The user defines the main directions and the number of subdivisions of the largest angle between two consecutive main directions. If more than two main directions are given, each interval between the main directions is subdivided such that the resulting small intervals have approximately the same size. This procedure defines a set of directions \mathbf{n}_j for which the extremal areas have to be determined.

For each direction \mathbf{n}_j the iso surface is subdivided into a series of equidistant parallel planes which are perpendicular to the current direction \mathbf{n}_j . On each of these planes all resulting orbits (cuts between the iso

surface and the plane) are determined. The algorithm can detect open and closed orbits. The open orbits are discarded. The set of resulting orbits as a function of plane index needs to be sorted into curves which connect pairs of related orbits on neighboring planes. This is in general a fuzzy task, since we do not have infinitesimal information. The code uses the gradient of the iso surface (Fermi velocity) to extrapolate the k -th orbit $O_{j,k}$ on plane \mathbf{n}_j onto plane \mathbf{n}_{j+1} . Then it goes through all actual orbits on plane \mathbf{n}_{j+1} and finds the one $O_{j+1,l}$, which most closely resembles the extrapolated orbit. If the deviation is less than a particular threshold (**area chain threshold**) $O_{j,k}$ and $O_{j+1,l}$ are considered to be connected in a continuous way and hence the two data points in the area curves are linked together. Of course the number of orbits on each plane can vary depending on the topology of the iso surface. Consequently, the resulting area versus plane index curves can contain a number of curves, which do not have to be connected and do not have to span the whole plane index interval. Fig. 13.5 shows an example of such area curves. It can be seen that some curves are connected and

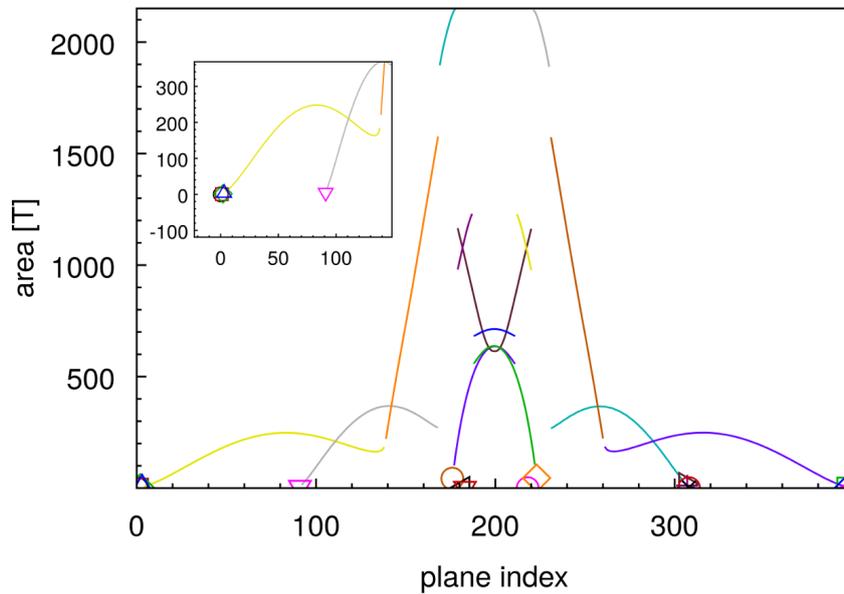


Figure 13.5: Example of areas vs. planes.

some are not. In general the connectivity is probably correct. It might be helpful to check particular cases by using the output (orbits) described later. One can also see that there are symbols in the graph. These denote curves of length one. There can be two reasons why they appear. The first is that there can be more than one symmetry related or periodically related orbit in one plane, even if one uses only the iso surface data from a single unit cell (which we do). The first, let's say, is linked to other orbits on other planes to form an area curve and the second is a left over, which would end up in another area chain if we used the iso surface data of more than one unit cells. This is actually not uncommon. The second reason for single data curves is a failure of the extrapolation-comparison algorithm described above. This can have two reasons. The first is that the gradient information of the iso surface is not good enough such that the linear extrapolation is not very good. The second is that linear extrapolation must fail close to points on the iso surface where we have second order extrema, like the tip of an egg. There the criterion for an acceptable extrapolation error will fail and hence the orbits do not get linked into a curve. However, the resulting extremal orbits, if they fall into such a section will have zero area and hence do not matter. This situation is shown in the inset of Fig. 13.5.

The explanation above should make clear that the dHvA stage depends critically on the accuracy of the iso surface stage.

The orbit sorting algorithm explained above is potentially slow. It becomes the slower the higher the accuracy of the iso surface is (it consists of more triangles) and the higher the number of planes and field intervals. To alleviate the pain, approximations are introduced. The first is a simple criterion which compares the areas and midpoints of the pairs of orbits. If the criterion is violated the pair of orbits is considered not to be related. The control parameter for this comparison is “**area radius factor**” (α) with default value $\alpha = 1$. It says that the midpoint of the two orbits must lay inside a circle of radius $\alpha \min(R_1, R_2)$. The second simplification has to do with the comparison of the extrapolated orbit to another orbit. In principle one has to compare each pair of points of both orbits to find which two points are closest to each other. Since an orbit is an ordered set of points it is enough to make a thorough search of closest points for the first point p_0 of orbit 1 only (result is q_{i_0} on orbit 2). The neighboring point p_1 on orbit 1 must have its closest partner q_{i_1} on orbit 2 somewhere close to q_{i_0} . We hence restrict the search of point partners to a certain subset of points on orbit 2, by only comparing the points $q_{i_0-\text{offset}} \dots q_{i_0+\text{offset}}$. This offset “**orbit comparison search offset**” is measured in percentage of the whole number of points with a default value of 0.2 or 20%. The last simplification is to not compare the whole set of points p_i of orbit 1 to establish linked pairs of orbits. It is sufficient to compare every n -th point only. This is measured by “**orbit sample portion**” with a default value 0.1 or 10%. This applies only if the number of points in the orbit exceeds 50.

After the orbits for a particular field direction have been sorted into curves (area vs plane index) the extrema of all these curves are determined. Since, the curves are represented in a discrete manner this is another fuzzy step. We chose to use 4 consecutive points to establish an extrema. These 4 points must have a monotonous piecewise derivative and must contain an extremum to be counted as such. If a curve has only 3 points in total the extremum is counted if the three point contain an extremum.

There are case cases in which the algorithm which links orbits into curves leads to wrong connections (mostly borderline cases). In generalt this leads to a jump in the now wrongly connected curve. It is however hard to tell if a large area difference in a 4-point interval is due to a jump or due to a large curvature. To rule out extreme cases a smoothness criterion is introduced (“**smoothness threshold**”, with a default value of 1) which measures the ratio of third to second derivative. The default value should basically switch the test off (owed to the imprecise nature of the criterion). Sometimes (if used, meaning $\text{threshold} < 1$) this measure discards proper extrema. However, an increase of the number of planes produces more data points on area curves and improves the extremum search, thus counteracting the wrong decision with respect to the discarded extrema. There is a natural limit to the increase of the number of planes which is related to the accuracy of the iso surface itself. The IS is formed of triangles. If two planes fall into the same triangle the resulting area curve starts to become a piecewise linear curve, which does not have good extremum conditions. Only experimenting with the number of planes and the accuracy of the IS will tell in each case what are good settings.

After the extrema are found for each field direction a last sorting step akin to the orbit sorting algo above will be employed to find out which extremal orbits of consecutive field directions should be connected into curves of extremal orbits. It uses extrapolation as before, only this time between planes of different inclination and distance. This sorting algorithm has the same fuzziness as the algo above. The resulting curves (extremal area vers field) are ment to be an orientation. If in doubt consult the visualization of the orbits an correct the curves by hand. You will find that certain parts of certain curves will be resolved better with an increase in allover accuracy.

In Fig. 13.6 results for the dHvA spectra for ZrB_2 are shown using different accuracy settings. It nicely illustrates the behaviour of the algorithms and their convergence properties. These pictures suggest that even with the highest settings used, it is not yet fully converged. Well, that’s how it is right now.

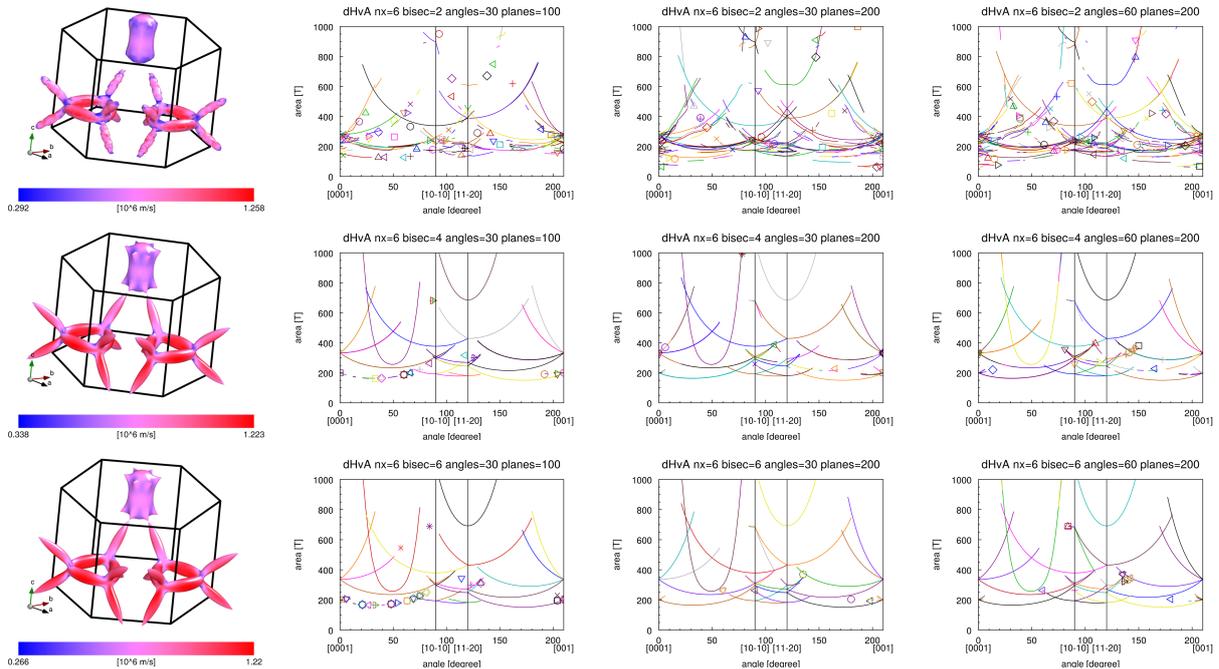


Figure 13.6: ZrB_2 dHvA spectra for various settings: nx is the initial mesh subdivision, $bisec$ is the number of bisections, $angles$ the maximum number of angles between two main directions and $planes$ the number of planes for each direction. dHvA accuracy increases from left to right and iso surface accuracy from top to bottom. One can clearly see that a bad iso surface produces several slightly different copies of the same final curve due to inaccuracies of the iso surface. One can also see how the higher accuracy increases the connectivity and even finds more extremal areas, of course for a higher cost.

After all, the explanations above should tell the user that the dHvA spectra will be a bit shaky in their precise appearance. This is owed to the complexity of the task itself.

The user can decide to treat specific iso surfaces by using the `Band indices/parts/spin` option (see below). The resulting set of treated iso surface files is written to the output as in

```
dHvA: isofiles
dHvA:           +iso_b00011_p0001_spin1
dHvA:           +iso_b00012_p0001_spin1
dHvA: fields
dHvA:           : [0.,0.,1.], "[0001]"
dHvA:           : [1.,0.,0.], "[10-10]"
dHvA:           : [0.866025403784,0.5,0.], "[11-20]"
dHvA:           : [0.,0.,1.], "[001]"
dHvA: nplane    : 50 , active range : 0 .. 49
dHvA: max nphi interv : 10 , active range : 5 .. 8
```

The main output of FDHVA are files called `+area_vs_angle_...` and `+mass_vs_angle_...`. The former contain the extremal areas as a function of an angular variable, which goes along the path through the main field directions. The latter contains the effective cyclotron masses which correspond to the curves in `+area_vs_angle_...`. These files contain comment lines stating the chain number of the corresponding curves. The data sets area in Tesla/mass in electron masses versus angle in degree. Each data line has a comment at

the end denoting the index of the corresponding angle for cross reference with other output and for debugging reasons. Note, that the mass curves are noisier than the area curves due to algorithmic peculiarities.

Additionally, the program writes a script file `area_vs_angle.cmd`, which can be loaded via

```
XFBP area_vs_angle.cmd
```

It contains read statements for all `+area_...` files, which were created during the latest FDHVA run. So, if you use the `Band indices/parts/spin` option to treat specific iso surfaces at a time the script does not necessarily contain all `+area_...` files in the directory.

For orientation and for checking the soundness of the results additional data are written into the directory `dHvAdata`.

For each iso surface which is treated according to the input settings a sub directory is created which is named according to the `b..._p...` suffixes of the iso surface files. The content of these `b..._p...` directories is deleted before each run, but only for currently treated iso surfaces. In those a subdirectory is created for each field direction, indexed by a continuous index called `iphi...`. This is the same index as written in the data line comments of `+area_...`. A list of field directions, angles and indices can be found in the output

```
maximum angle between main directions: 90
B[ 0]= [ 0.00000 0.00000 1.00000] dphi= 0
B[ 1]= [ 0.15643 0.00000 0.98769] dphi= 9
B[ 2]= [ 0.30902 0.00000 0.95106] dphi= 18
B[ 3]= [ 0.45399 0.00000 0.89101] dphi= 27
B[ 4]= [ 0.58779 0.00000 0.80902] dphi= 36
B[ 5]= [ 0.70711 0.00000 0.70711] dphi= 45
B[ 6]= [ 0.80902 0.00000 0.58779] dphi= 54
...
```

The index is the number in the square brackets. The angular variable is `dphi`. For a default set of output options these `iphi...` directories contain two files `areas` (and `masses`) with the area curves versus plane index of the corresponding field direction. These can be used to check if a certain extremal area in the final result really looks like an extrema.

If the option `output_orbits` is set additionally data files for all orbits are written. The names are `orbit_plane..._chain...`. The format is legacy Vtk. They should be readable by software, which reads vtk files, but also by XFLO. Since these files are not specific to the Fermi surface mode of XFLO we need a special calling strategy:

```
XFLO -fs dHvAdata/b00011_p0001_spin1/iphi00012/orbit_plane*_chain7*
```

will load all orbits of planes which belong to chain 7 of angle 12 of iso surface band 122 part 1 spin 1. Alternatively, if a file `=.xef` with proper options exists you can write

```
XFLO =.xef dHvAdata/b00011_p0001_spin1/iphi00012/orbit_plane*_chain7*
```

or together with the iso surface

```
XFLO -fs dHvAdata/b00011_p0001_spin1/iphi00012/orbit_plane*_chain7* -raw +iso_b00011_p0001_spin1
```

Btw, this does not mean that XFLO can read any legacy Vtk file ;-)

A more useful output option (which is checked by default) is `output_extremal_orbits`, which triggers the output of the extremal orbits `extrorbit_chain..._iphi..._vtk` in the `b..._p...` sub-directories. Load them together with the iso surface via

```
XFPLO -fs dHvAdata/b00011_p0001_spin1/extrorbit_chain00000_*.vtk -raw +iso_b00011_p0001_spin1
```

You might notice that the extremal orbits seem to be spaced a little uneven at times. This is due to a technical aspect, which makes us chose the orbit of the plane closest to the extremum for output. Consequently, the orbits are off by half a plane distance in the worst case. The visualization of the extremal orbits is helpful for identifying the origin of certain branches of the dHvA spectrum.

A few warnings:

- Don't try to load the orbits from within their respective sub directories. XFPL0 needs to find `=.in`.
- If you are in an `iphi...` subdirectory while it gets deleted by FDHVA it's files might no longer be accessible after the fact. Change back out and in of the directory to remedy this.
- The `+area_...` and `+mass_...` files are never deleted since the user should be able to investigate one iso surface sheet at a time. The user must take care of old files!

13.2.1 Input options

The input which controls the dHvA stage is edited via the second part of the dHvA submenu of FEDIT shown in Fig. 13.7.

Number of fields Sets the number of main field directions.

No. Label field For each main field direction give a label for the field and the actual direction (which does not need to be normalized).

Angle subdiv Define the number of angular subdivisions between the two consecutive main field directions with the largest angle between them.

Range (Debugging option) This can be used to narrow the number of resulting angles (field directions) to a certian interval. Negative numbers mean ignore.

No of planes Sets the number of planes with which to slice the iso surface for each field direction.

range (Debugging option) This can be used to narrow the number of planes to a certian interval. Negative numbers mean ignore.

Band indices/parts/spin By default the dHvA spectrum is calculated for each `+iso_b...` files present in the directory. If these values are set, only specific iso surfaces are treated.

Numerics: In general we believe that the numerical control parameters (explained above in detail) are chosen well with their default values.

Area chain threshold The maximum error allowed when linking two orbits into a chain.

Orbit comparison search offset When comparing two orbits only search a certain portion of the second orbit to find a matching point. (Explained in detail above)

Orbit sample portion Only try to match a certain portion of orbit 1, when sorting into linked curves. (Explained in detail above)

Area radius factor When linking orbits into chains exclude all pairs of orbits, whose midpoint's distance is larger than "Area radius factor" times the minimum of the two orbit's radii. (Explained in detail above)

```

DHVA
e (X)it (/) Search (H)elp
[...]
Energy shift
(P) number of shift terms : 0
-----
No. weight-index factor
-----
DHVA
(N)umber of fields : 4
-----
No. Label field
-----
(1) : [0001] : 0 0 1
(2) : [10-10] : 1 0 0
(3) : [11-20] : 0.866025403784 0.5 0
(4) : [001] : 0. 0. 1.

An (G)le subdiv : 30 (Y1) range : -1 .. -2
No o (F) planes : 400 (Y2) range : -1 .. -2

B (A)nd indices : par (T)s : 1 sp (I)n : both

Numerics
(M1) area chain threshold : 0.1
(M2) orbit comparison search offse : 0.2
(M3) orbit sample portion : 0.1
(M4) area radius factor : 1.
(M5) smoothness threshold : 3.

(-) Output Options
show_non_closed : [ ] output_orbits : [ ]
output_extremal_orbits : [X]

(=) Debug Options
determine_area_chain_extrema : [ ] sort_area_chains : [ ]
sort_extrema_chains : [ ] extend_periodic_area_chains : [ ]
collect_orbits : [ ] calc_orbit : [ ]
run : [X]

STATUS: OK (18.00-52:M-CPA)

```

Figure 13.7: Second part of the dHvA sub menu in FEDIT.

Smoothness threshold When searching for extrema exclude extrema for which the local 4-point interval has a smoothness not smaller than this threshold. Zero excludes all extrema and one allows all extrema. (Explained in more detail above)

Output Options: For an in depth analysis of the results it might be helpful to create additional output files.

show_non_closed If `output_orbits` is checked not only the closed orbits are written to files but also the open orbits.

output_orbits Write the orbits of all planes into files in `dHvAdata/b...p...spin.../iphi.../`.

`output_extremal_orbits` Checked by default. Triggers output of all extremal orbits for all treated is surfaces in dHvAdata/b...p...spin.../.

`Debug Options`: These options trigger debug output and are of lesser importance to the user. Hence, we will not explain them.

`determine_area_chain_extrema`

`sort_area_chains`

`sort_extrema_chains`

`extend_periodic_area_chains`

`collect_orbits`

`calc_orbit`

`run`



Chapter 14

Topological insulators

For topological insulators the Z_2 invariants $\nu_0; (\nu_1\nu_2\nu_3)$ can be calculated. There is a FEDIT submenu with a corresponding help screen to explain the input. In the newer version it is possible to calculate the invariants for systems with and without inversion symmetry. The latter uses Wannier centers to find the invariants. This is a rather involved topic, which requires intent user participation. The essentials are explained in the context of the PYFPLO module elsewhere [../pyfplo/pyfplo.pdf](#) since internally the same code is used. Note, that if calculated by FPLO the details of the results differ, since an all-orbital Wannier basis is constructed, while in PYFPLO a user defined reduced basis Wannier model is constructed. The topology should not change. However, the FPLO-version of the algorithm is naturally slower and usually shows more Wannier centers (semi core and other fully occupied orbitals, omitted in the reduced Wannier model). If surface states are desired it is always better to first construct a Wannier model and to use PYFPLO. This is just faster at the end.



Chapter 15

Band Unfolding

15.1 Citations

When a citation is need it would be [1, 6, 12]. The second is the idea by Wei Ku and the third the first FPLO implementation/application.

15.1.1 Copy right note

The content of this report cannot be used in another publication without contacting the FPLO authors first.

15.2 Introduction

15.2.1 Original idea

The idea is due to Wei Ku [6]. Let's assume that we have a normal cell (NC) and a super cell (SC) derived from it. The NC contains N_s atoms $\mathbf{s}_i \in [1, N_s]$ and has N_r lattice vectors \mathbf{r} in a defined Born von Karman torus. The SC contains N_S atoms $\mathbf{S}_i \in [1, N_S]$ and has N_R lattice vectors \mathbf{R} in the **same** BvK torus. The SC is a "multiple" of NC, meaning $N_S = fN_s$, $f \in \mathbb{N}$ and because of the same BvK torus $N_R = \frac{1}{f}N_r$, such that both tori contain the same number of sites $N_s N_r = N_S N_R$. The KS states of both cells fulfill their respective Bloch theorem

$$\begin{aligned}\Psi_{\mathbf{k}N}(\mathbf{r} - \mathbf{p}) &= \Psi_{\mathbf{K}N}(\mathbf{r}) e^{-i\mathbf{k}\mathbf{p}} \\ \Psi_{\mathbf{K}N}(\mathbf{r} - \mathbf{P}) &= \Psi_{\mathbf{K}N}(\mathbf{r}) e^{-i\mathbf{K}\mathbf{P}}\end{aligned}$$

where we denote reciprocal lattice vectors in the NC Brillouin zone (bz) with \mathbf{k} and reciprocal lattice vectors in the SC Brillouin zone (BZ) with \mathbf{K} . The Bloch spectral density is defined as

$$\hat{A}(\omega) = -\frac{1}{\pi} \text{Im} \hat{G}(\omega)$$

with the single particle Green's function

$$\hat{G}(\omega) = \frac{1}{\omega^+ - \hat{H}}$$

and reads in the SC KS basis

$$\begin{aligned}A_{\overline{\mathbf{K}N}, \mathbf{K}N}(\omega) &= \langle \overline{\mathbf{K}N} | \hat{A} | \mathbf{K}N \rangle \\ &= \delta_{\overline{\mathbf{K}}\mathbf{K}} \delta_{\overline{N}N} \delta(\omega - \varepsilon_{\mathbf{K}N})\end{aligned}$$

It is diagonal in the SC quantum numbers. The full Green's function can be written as

$$\hat{G}^{-1} = \hat{G}_0^{-1} - \hat{V}$$

where G_0 is the Green's function of the NC and V is the perturbing potential, which makes the difference between the exactly duplicated NC and the actual SC. Here, we already assume that only the potential is different, meaning that all atoms are exactly the same, just moved around a bit. We will discuss this later. Of course, if V is weak, one can argue that the NC KS states $\Psi_{\mathbf{k}n}$ are also a good basis to express the Bloch spectral density. When done so, A will no longer be diagonal in the \mathbf{k} indices, since the NC Bloch symmetry is broken in SC. However, focusing only on the diagonal elements one get's an approximation for A , which will have approximately the Bloch symmetry of the NC and hence $A_{\mathbf{k}n, \mathbf{k}n} = \langle \mathbf{k}n | \hat{A} | \mathbf{k}n \rangle$ will be the unfolded Bloch spectral density. Of course, $\Psi_{\mathbf{k}n}$ can only be a basis of SC, if SC is a "multiple" of NC (no atom substitution, see discussion in Section 15.3).

15.2.2 The local orbital connection

Now, one can express the SC $\Psi_{\mathbf{K}N}$ in terms of Wannier functions (WFs) and claim that the WFs are the same in NC. Then one can transfer the WFs from SC to NC and build the KS functions in both cells from the same basis. This allows to express everything in terms of these WFs. In FPLO the local basis forms a complete WF basis, if Löwdin orthogonalized. (This orthogonalization leads to a new basis $\tilde{\Phi} = \Phi S^{-\frac{1}{2}}$, which has the property of being the one "closest" to the non-orthogonal basis. Hence, an L-ortho FPLO basis is very localized and is a WF basis. To avoid unnessecary complications we work in our non-ortho FPLO basis and use gross-projection, when needed ($\frac{1}{2}(S * + * S)$). Under the assumption that no atom substitution has taken place in SC we can do the algebra and calculate the following. The KS states are expressed via Bloch sums of WFs/local orbitals

$$\Phi_{S\mu}^{\mathbf{K}} = \frac{1}{\sqrt{N_R}} \sum_{\mathbf{R}} \Phi_{RS\mu} e^{i\mathbf{K}(\mathbf{R}+\mathbf{S})}$$

where $\Phi_{RS\mu}$ is the μ orbital at site \mathbf{S} in cell \mathbf{R} . The KS-state is a simple linear combination of these Bloch sums

$$\Psi_{\mathbf{K}N} = \sum_{S\mu} \Phi_{S\mu}^{\mathbf{K}} C_{S\mu, N}^{\mathbf{K}}$$

Hence,

$$\begin{aligned} \hat{A} &= \sum_{\mathbf{K}N} \Psi_{\mathbf{K}N} \delta(\omega - \varepsilon_{\mathbf{K}N}) \Psi_{\mathbf{K}N}^{\dagger} \\ &= \sum_{\mathbf{K}N} \sum_{S\mu, \bar{S}\bar{\mu}} \Phi_{S\mu}^{\mathbf{K}} n_{S\mu, \bar{S}\bar{\mu}}^{\mathbf{K}N}(\omega) \Phi_{\bar{S}\bar{\mu}}^{\mathbf{K}} \end{aligned}$$

which introduces the DOS weight matrix in the local basis

$$n_{S\mu, \bar{S}\bar{\mu}}^{\mathbf{K}N}(\omega) = C_{S\mu, N}^{\mathbf{K}} \delta(\omega - \varepsilon_{\mathbf{K}N}) C_{\bar{S}\bar{\mu}, N}^{\mathbf{K}*}$$

Now, we map the sites of the SC onto the sites of NC. The SC consists of f "copies" of NC, translated by NC lattice vectors \mathbf{r}_j , $j \in [1, f]$. For every site \mathbf{S} there is a site \mathbf{s} in NC, which is in the NC cell translated by \mathbf{r}_j . The SC lattice vector and the translation \mathbf{r}_j give a NC lattice vector

$$\mathbf{r} = \mathbf{R} + \mathbf{r}_j$$

and the site mapping can be written by labeling the SC sites according to their construction out of NC sites and cells (cf. Figure 15.1)

$$\mathbf{s} + \mathbf{r}_j = \mathbf{S}_{sj}.$$

Consequently, absolute site vectors can be written

$$\mathbf{r} + \mathbf{s} = \mathbf{R} + \mathbf{r}_j + \mathbf{s} = \mathbf{R} + \mathbf{S}_{sj}$$

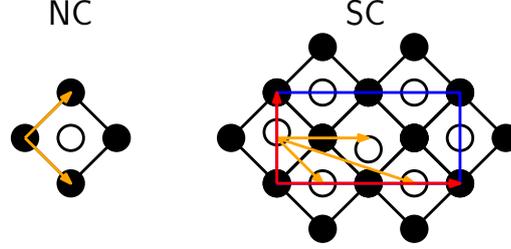


Figure 15.1: Cell mapping: The SC as the “fourfold” of the NC. Some of the open atoms are displaced against ideal NC positions. The small vectors are NC lattice vectors and show how the 4 SC open atoms are mapped onto their one NC equivalent.

If we now neglect eventual atom displacements we can form SC Bloch sums out of NC Bloch sums (but at the SC \mathbf{k} -vector)

$$\begin{aligned} \Phi_{[NC]s}^{\mathbf{K}} &= \frac{1}{\sqrt{N_r}} \sum_r \Phi_{rs\mu} e^{i\mathbf{K}(\mathbf{r}+\mathbf{s})} \\ &= \frac{\sqrt{N_R}}{\sqrt{N_r}} \left(\frac{1}{\sqrt{N_R}} \sum_{Rj} \Phi_{Rr_j, s\mu} e^{i\mathbf{K}(\mathbf{R}+\mathbf{r}_j+\mathbf{s})} \right) \\ &= \frac{1}{\sqrt{f}} \left(\frac{1}{\sqrt{N_R}} \sum_{Rj} \Phi_{RS_{sj}\mu} e^{i\mathbf{K}(\mathbf{R}+\mathbf{S}_{sj})} \right) \\ \Phi_{[NC]s}^{\mathbf{K}} &= \frac{1}{\sqrt{f}} \sum_j \Phi_{[SC]S_{sj}}^{\mathbf{K}} \end{aligned} \quad (15.1)$$

The NC Bloch sum is a contraction of the SC Bloch sums. The KS states behave as

$$\begin{aligned} \Psi_N^{\mathbf{K}} &= \sum_{S\mu} \Phi_{S\mu}^{\mathbf{K}} C_{S\mu, N}^{\mathbf{K}} \\ &= \sum_{sj\mu} \Phi_{S_{sj}\mu}^{\mathbf{K}} C_{S_{sj}\mu, N}^{\mathbf{K}} \end{aligned}$$

Now, the coefficients would fulfill the translational symmetry

$$C_{S_{sj}} = C_{r_j s} = C_s U$$

if \mathbf{r}_j is strictly a lattice vector. (U denotes a possible unitary transformation, which does not figure in the following, where we reverse the argumentation). Thus, if the NC translational symmetry were true, we could replace

$$C_{S_{sj}} = \frac{1}{f_s} \sum_{j=1}^{f_s} C_{S_{sj}}$$

Let's do that in the KS state and use Eq. (15.1)

$$\begin{aligned}
\Psi_N^K &\approx \sum_{sj\mu} \Phi_{S_{sj\mu}}^K \frac{1}{f_s} \sum_{i=1}^{f_s} C_{S_{si\mu},N}^K \\
&\approx \sqrt{f} \sum_{s\mu} \Phi_{[NC]s\mu}^K \frac{1}{f_s} \sum_{i=1}^{f_s} C_{S_{si\mu},N}^K \\
&= \frac{\sqrt{f}}{f_s} \sum_{s\mu, i=1}^{f_s} \Phi_{[NC]s\mu}^K C_{S_{si\mu},N}^K
\end{aligned}$$

Hence, we get the coefficients in the NC Bloch sum representation

$$C_{[NC]s\mu,N}^K = \frac{\sqrt{f}}{f_s} \sum_{j=1}^{f_s} C_{S_{sj\mu},N}^K \quad (15.2)$$

and

$$\Psi_N^K \approx \sum_{s\mu} \Phi_{[NC]s\mu}^K C_{[NC]s\mu,N}^K$$

Now, we can write the Bloch spectral density in terms of these approximate Bloch sums

$$\hat{A} \approx \sum_{KN} \sum_{s\mu, \bar{s}\bar{\mu}} \Phi_{[NC]s\mu}^K \left[\frac{f}{f_s f_{\bar{s}}} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_{\bar{s}}} n_{S_{sj\mu}, \bar{S}_{\bar{s}\bar{j}\bar{\mu}}}^{KN}(\omega) \right] \Phi_{[NC]\bar{s}\bar{\mu}}^K \quad (15.3)$$

The un-approximated Bloch spectral function fulfills

$$\int_{-\infty}^{\infty} A_{KN}(\omega) = \int_{-\infty}^{\infty} \text{Tr} \hat{A}_{KN}(\omega) = 1$$

with $\hat{A}_{KN} = |\mathbf{KN}\rangle \delta(\omega - \varepsilon_{KN}) \langle \mathbf{KN}|$. The Bloch sums yield the overlap matrix via

$$S_{\bar{s}\bar{\mu}, s\mu}^K = \langle \Phi_{\bar{s}\bar{\mu}}^K | \Phi_{s\mu}^K \rangle$$

So, we basically get $\int A = \int \text{Tr} n^K(\omega) S^K$, which is nothing but the normalization condition $C^+ S C = 1$ for non-orthogonal eigenvalue problems. Using Eq. (15.2) it is clear that Eq. (15.3) is properly normalized if the NC Bloch symmetry is exact. (Note, that then $f_s = f$.) If we introduce a symmetrized gross projected method, where we use contragradient Bloch sums $\tilde{\Phi}^K = \Phi^K (S^K)^{-1}$, $\langle \Phi | \tilde{\Phi} \rangle = 1$ on one side of A we get the expression

$$\hat{A} = \frac{1}{2} \left(\Phi n S \tilde{\Phi}^+ + \tilde{\Phi} S n \Phi^+ \right)$$

and the trace give

$$\text{Tr} \underline{A} = \frac{1}{2} \text{Tr} (nS + Sn)$$

Hence, the diagonal part of $\underline{A} = \frac{1}{2} (nS + Sn)$ gives the unfolded weights and is normalized ($\text{Tr} \underline{A} = 1$), if the NC Bloch symmetry is strictly true. For cases, where atoms got substituted, we have to make a modification to Eq. (15.1). The f should be the actual multiplicity of the NC site s under consideration $f = f_s$. This is a definition, which is consistent with any possible choice of NC and also with the cases of partial unfolding due to substitutions. In any case the normalization is proper. Consider the case of perfect NC Bloch symmetry.

Then $n_{[SC]S_{sj}} = \frac{1}{f} n_{[NC]_s}$. Hence (omitting S for simplicity) the diagonal part of Eq. (15.3) reads

$$\begin{aligned} A_s &= \frac{f}{f_s f_s} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_s} n_{[SC]S_{sj}S_{s\bar{j}}} \\ &= \frac{f}{f_s f_s} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_s} \frac{1}{f} n_{[NC]_{ss}} \\ &= n_{[NC]_{ss}} \end{aligned}$$

And normalization becomes

$$\sum_s \text{Tr}_\mu A_{s\mu} = 1$$

which is fulfilled since it is for the NC expression. If we now have to partially unfold because out of f sites, which would backfold to a single NC site, several (let's say m_s) are occupied by a different atom then the $f_s = f - m_s$ other sites, we cannot fully contract over all f sites. But we can contract over $f_s = f - m_s$ sites and leave the m_s other atoms uncontracted (means not unfolded). This would restrict the contraction sum to $f_s \neq f$. Let's assume that still approximately $n_{[SC]S_{sj}} = \frac{1}{f} n_{[NC]_s}$ (just to discuss the normalization) then we get, letting the first parameter $f = F$ be chosen later

$$\begin{aligned} A_s &= \frac{F}{f_s f_s} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_s} n_{[SC]S_{sj}S_{s\bar{j}}} \\ &= \frac{F}{f} n_{[NC]_{ss}} \end{aligned}$$

Besides this we have the m_s un-unfolded weights of the sites s_λ , which have substituted atoms

$$\begin{aligned} A_{s_\lambda} &= \frac{1}{f} n_{[NC]_{s_\lambda s_\lambda}}, \quad \lambda \in [1, m_s] \\ &\approx \frac{1}{f} n_{[NC]_{ss}}, \quad (\text{if perfect}) \end{aligned}$$

The normalization reads

$$A_s + \sum_{\lambda=1}^{m_s} A_{s_\lambda} = \frac{F + m_s}{f} n_{[NC]_{ss}}$$

and it becomes clear that $F = f_s$ gives normalization when summed over all sites. Hence our final definition is

$$w_{s\mu}^{\mathbf{KN}}(\omega) = \frac{1}{f_s} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_s} n_{S_{sj}\mu S_{s\bar{j}}\mu}^{\mathbf{KN}}(\omega) \quad (15.4)$$

Or in gross projection and dropping the delta function, which was a placeholder for the energy dependence

$$\begin{aligned} w_{s\mu}^{\mathbf{KN}} &= \frac{1}{f_s} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_s} \frac{1}{2} \left(C_{S_{sj}\mu, N}^{\mathbf{K}} [C^+ S]_{N, S_{s\bar{j}}\mu}^{\mathbf{K}} + [SC]_{S_{sj}\mu, N}^{\mathbf{K}} C_{S_{s\bar{j}}\mu, N}^{\mathbf{K}*} \right) \\ &= \frac{1}{f_s} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_s} \frac{1}{2} \left(C_{S_{sj}\mu, N}^{\mathbf{K}} [C^+ S]_{N, S_{s\bar{j}}\mu}^{\mathbf{K}} + [C^+ S]_{N, S_{s\bar{j}}\mu}^{\mathbf{K}*} C_{S_{sj}\mu, N}^{\mathbf{K}*} \right), \quad j \leftrightarrow \bar{j}, \text{ in second term} \\ &= \frac{1}{f_s} \sum_{j=1}^{f_s} \sum_{\bar{j}=1}^{f_s} \text{Re} \left(C_{S_{sj}\mu, N}^{\mathbf{K}} [C^+ S]_{N, S_{s\bar{j}}\mu}^{\mathbf{K}} \right) \end{aligned}$$

Net projection will be discussed below.

15.2.3 Alternative considerations

First let's defined the normal cell problem and derive some symmetries. The orbitals $\Phi_{r s \mu}$ form Bloch sums

$$\Phi_{s \mu}^{\mathbf{k}} = \frac{1}{\sqrt{N_r}} \sum_r \Phi_{r s \mu} e^{i \mathbf{k}(\mathbf{r}+\mathbf{s})}$$

and wave functions

$$\Psi_n^{\mathbf{k}} = \sum_{s \mu} \Phi_{s \mu}^{\mathbf{k}} C_{s \mu, n}^{\mathbf{k}}$$

where C is determined by (overlap ignored, since it does not change the results derived here)

$$H^{\mathbf{k}} = (\Phi^{\mathbf{k}} | H | \Phi^{\mathbf{k}})$$

$$H^{\mathbf{k}} C^{\mathbf{k}} = C^{\mathbf{k}} \varepsilon^{\mathbf{k}}$$

At a shifted \mathbf{k} -vector we get

$$\begin{aligned} \Phi_s^{\mathbf{k}+\mathbf{g}} &= \Phi_{s \mu}^{\mathbf{k}} e^{i \mathbf{g}(\mathbf{r}+\mathbf{s})} \\ &= \Phi_{s \mu}^{\mathbf{k}} e^{i \mathbf{g} \mathbf{s}} \end{aligned}$$

and hence

$$H_{\bar{s} \bar{s}}^{\mathbf{k}+\mathbf{g}} = e^{-i \mathbf{g} \bar{\mathbf{s}}} H_{\bar{s} \bar{s}}^{\mathbf{k}} e^{i \mathbf{g} \bar{\mathbf{s}}}$$

$$H^{\mathbf{k}+\mathbf{g}} C^{\mathbf{k}+\mathbf{g}} = C^{\mathbf{k}+\mathbf{g}} \varepsilon^{\mathbf{k}+\mathbf{g}}$$

$$H^{\mathbf{k}} (e^{i \mathbf{g} \mathbf{s}} C^{\mathbf{k}+\mathbf{g}}) = (e^{i \mathbf{g} \mathbf{s}} C^{\mathbf{k}+\mathbf{g}}) \varepsilon^{\mathbf{k}+\mathbf{g}}$$

which says that the expression in parentheses is also an eigenvector for \mathbf{k} and hence

$$\varepsilon^{\mathbf{k}+\mathbf{g}} = \varepsilon^{\mathbf{k}}$$

$$\boxed{C^{\mathbf{k}+\mathbf{g}} = e^{-i \mathbf{g} \mathbf{s}} C^{\mathbf{k}} U^{\mathbf{k}}} \quad (15.5)$$

where U mixes only states in degenerate subspaces (degenerate bands along symmetry lines/planes or at band crossings). Now, one can argue that there is always a way to adjust the phases (gauge freedom) of an actually calculated $C^{\mathbf{k}+\mathbf{g}}$ (solution to the eigenvalue problem) such that the result of a shift by \mathbf{g} is at most a reordering of band indices, which allows to set $U = 1$ in our considerations. In essence we are saying that the set over all band indices $\bigcup_n \Psi_n^{\mathbf{k}+\mathbf{g}}$ equals $\bigcup_n \Psi_n^{\mathbf{k}}$. We have periodicity of the sets of bands with respect to reciprocal translations. This is strictly only true for the sets, which happen to form continuous and periodic functions of \mathbf{k} (yes functions of sets!). If an individual band is followed smoothly (e.g. by the $\mathbf{k} \cdot \mathbf{p}$ -method) the resulting $\Psi_n^{\mathbf{k}+\mathbf{g}}$ can become another band $\Psi_{n'}^{\mathbf{k}}$.

Now, we describe the same system by introducing a super cell with f copies of the original cell. This leads to new lattice vectors \mathbf{R} and sites $\mathbf{S}_{sj} = \mathbf{r}_j + \mathbf{s}$, which are formed by shifting the original sites \mathbf{s} via a set of f original lattice vectors \mathbf{r}_j , $j \in [1, f]$. Of course, the set of normal cell lattice vectors is obtained from $\{\mathbf{r}\} = \bigcup_{j=1}^f \{\mathbf{R} + \mathbf{r}_j\}$, which also leads to $\sum_{R_j} F(\mathbf{R} + \mathbf{r}_j) = \sum_r F(\mathbf{r})$. We also have to require that the BvK torus in both descriptions has the same volume or that the number of normal cell lattice vectors is f -times the number of super cell lattice vectors: $N_r = f N_R$. Now, we can reformulate the problem in this super cell description by forming Bloch sums with proper super cell translational symmetry

$$\Phi_{[SC]S_{sj}\mu}^{\mathbf{k}} = \frac{1}{\sqrt{N_R}} \sum_R \Phi_{RS_{sj}\mu} e^{i \mathbf{k}(\mathbf{R}+\mathbf{S}_{js})}$$

The corresponding coefficients are $C_{[SC]S_{sj}\mu,n}^{\mathbf{k}}$.

This description cannot lead to a different result compared to the normal cell description, since we only changed the artificial choice of a supercell. However, by increasing the matrix size (number of sites per unit cell) we increased the number of eigen solutions or bands per \mathbf{k} -point. In order to get the same number of physically different solutions the size of the reciprocal unit cell must be f -times smaller than the NC reciprocal unit cell. Well, as everyone knows the excess solutions are bands, which got backfolded into this smaller reciprocal cell. To understand this mathematically it suffices to note that we already showed that in every unit cell the sets of bands form periodic and continuous functions. Hence, the SC wave functions are periodic with respect to translations by reciprocal lattice vectors \mathbf{G} : $\bigcup_n \Psi_n^{\mathbf{k}+\mathbf{G}} = \bigcup_n \Psi_n^{\mathbf{k}}$. Let's construct the $f \cdot m$ SC solutions out of the m NC solutions. The first m solutions are just identical to the NC solutions $\bigcup_{n=1}^m \Psi_{[SC]n}^{\mathbf{k}} = \bigcup_{n=1}^m \Psi_{[NC]n}^{\mathbf{k}}$. Now, for every reciprocal vector \mathbf{G} the set $\bigcup_{n=1}^m \Psi_{[NC]n}^{\mathbf{k}+\mathbf{G}}$ is either equivalent to $\bigcup_{n=1}^m \Psi_{[NC]n}^{\mathbf{k}}$ (\mathbf{G} is a NC vector \mathbf{g}) or forms a new backfolded set in which case there are exactly $f - 1$ different additional sets. We identify $f - 1$ representative vectors $\mathbf{G}_l \notin \{\mathbf{g}\}$, $l \in [1, f - 1]$ and $\mathbf{G}_0 = 0$ and define the set of $f \cdot m$ SC solutions via $\bigcup_{n=1}^m \Psi_{[SC]n+lm}^{\mathbf{k}} = \bigcup_{n=1}^m \Psi_{[NC]n}^{\mathbf{k}+\mathbf{G}_l}$, $\forall l \in [0, f - 1]$. This is just fancy talk for all the backfolding. Beware, that in a real calculations all coefficients C contain random phase factors from the eigenvalue solver, which means that the construction above deviates in practice from the actual solutions by phases and unitary mixing of degenerate bands.

However, now we can go a step further by identifying equivalences between coefficients. Using the construction of the SC sites discussed above we can write

$$\begin{aligned} \sum_S \Phi_{[SC]S_{sj}\mu}^{\mathbf{k}} &= \frac{1}{\sqrt{N_R}} \sum_{RS} \Phi_{RS_{js}\mu} e^{i\mathbf{k}(\mathbf{R}+\mathbf{S}_{js})} \\ &= \frac{1}{\sqrt{N_R}} \sum_{Rj_s} \Phi_{Rr_j s\mu} e^{i\mathbf{k}(\mathbf{R}+\mathbf{s}+\mathbf{r}_j)} \\ &= \frac{\sqrt{f}}{\sqrt{N_r}} \sum_{rs} \Phi_{rs\mu} e^{i\mathbf{k}(\mathbf{r}+\mathbf{s})} \end{aligned}$$

which leads to the SC expression

$$\begin{aligned} \Psi_{[SC]n+lm}^{\mathbf{k}} &= \sum_{S\mu} \Phi_{[SC]S\mu}^{\mathbf{k}} C_{[SC]S\mu,n+lm}^{\mathbf{k}} \\ &= \frac{1}{\sqrt{N_R}} \sum_{RS\mu} \Phi_{RS\mu} e^{i\mathbf{k}(\mathbf{R}+\mathbf{S})} C_{[SC]S\mu,n+lm}^{\mathbf{k}} \\ &= \frac{\sqrt{f}}{\sqrt{N_r}} \sum_{Rj_s\mu} \Phi_{Rr_j s\mu} e^{i\mathbf{k}(\mathbf{R}+\mathbf{r}_j+\mathbf{s})} C_{[SC]S_{js}\mu,n+lm}^{\mathbf{k}} \end{aligned}$$

and the NC expression, whose equivalence via backfolding we established in the argument above.

$$\begin{aligned} \Psi_{[SC]n+lm}^{\mathbf{k}} &= \Psi_{[NC]n}^{\mathbf{k}+\mathbf{G}_l} \\ &= \sum_{s\mu} \Phi_{[NC]s\mu}^{\mathbf{k}+\mathbf{G}_l} C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \\ &= \frac{1}{\sqrt{N_r}} \sum_{rs\mu} \Phi_{rs\mu} e^{i(\mathbf{k}+\mathbf{G}_l)(\mathbf{r}+\mathbf{s})} C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \\ &= \frac{1}{\sqrt{N_r}} \sum_{Rj_s\mu} \Phi_{Rr_j s\mu} e^{i(\mathbf{k}+\mathbf{G}_l)(\mathbf{R}+\mathbf{r}_j+\mathbf{s})} C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \\ &= \frac{1}{\sqrt{N_r}} \sum_{Rj_s\mu} \Phi_{Rr_j s\mu} e^{i\mathbf{k}(\mathbf{R}+\mathbf{r}_j+\mathbf{s})} e^{i\mathbf{G}_l(\mathbf{r}_j+\mathbf{s})} C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \end{aligned}$$

By comparison with the SC expression we identify

$$\begin{aligned}
 C_{[SC]S_{j_s}\mu,n+lm}^{\mathbf{k}} &= \frac{1}{\sqrt{f}} e^{i\mathbf{G}_l \mathbf{S}_{s_j}} C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \\
 &= \frac{1}{\sqrt{f}} e^{i\mathbf{G}_l \mathbf{r}_j} e^{i\mathbf{G}_l \mathbf{s}} C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \quad (15.6)
 \end{aligned}$$

which is (up to unitary mixing, which we neglected) the exact mapping between the solutions of the two equivalent descriptions. Now, we average the SC coefficients over the equivalent sites with respect to NC periodicity (j -sum with f_s terms).

$$\frac{1}{f_s} \sum_j C_{[SC]S_{j_s}\mu,n+lm}^{\mathbf{k}} = \frac{1}{\sqrt{f}} \frac{1}{f_s} \left(\sum_j e^{i\mathbf{G}_l \mathbf{r}_j} \right) e^{i\mathbf{G}_l \mathbf{s}} C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l}$$

The sum in parentheses runs over all NC lattice vectors \mathbf{r}_j , which are needed to make the whole NC lattice from the SC lattice vectors \mathbf{R} . We can introduce a basis in the real space lattice via

$$\begin{aligned}
 \mathbf{R} &= \sum_{I=1}^3 R_I \mathbf{A}_I \\
 \mathbf{r} &= \sum r_I \mathbf{a}_I
 \end{aligned}$$

where in order for the SC to be a commensurate multiple of the NC the relation $\mathbf{A}_I = \mathbf{a}_J M_{JI}$, $M_{JI} \in \mathbb{N}$ must hold. The reciprocal lattice then also has a basis

$$\begin{aligned}
 \mathbf{G} &= \sum G_I \mathbf{B}_I \\
 \mathbf{g} &= \sum g_I \mathbf{b}_I
 \end{aligned}$$

with the defining relation of the reciprocal basis $\mathbf{B}_I \mathbf{A}_J = \mathbf{b}_I \mathbf{a}_J = 2\pi \delta_{IJ}$. Hence $\mathbf{B}_I \mathbf{a}_K M_{KJ} = 2\pi \delta_{IJ}$ or $\mathbf{B}_I \mathbf{a}_J = 2\pi (M^{-1})_{IJ}$

$$\begin{aligned}
 \mathbf{G}_l \mathbf{r}_j &= \sum_I G_{l,I} \mathbf{B}_I \cdot \sum_J r_{j,J} \mathbf{a}_J \\
 &= \sum_{IJ} G_{l,I} r_{j,J} \mathbf{B}_I \cdot \mathbf{a}_J \\
 &= 2\pi \sum_{IJ} G_{l,I} r_{j,J} (M^{-1})_{IJ}
 \end{aligned}$$

The inverse of an integer matrix M is a matrix of rational numbers. The coefficients $G_{l,I} r_{j,J}$ in the equation above are integer in such a way that if \mathbf{G}_l were a reciprocal lattice vector \mathbf{g} of the NC $G_{l,I} (M^{-1})_{IJ} \in \mathbb{N}$ must be an integer because $\mathbf{g}\mathbf{r} = 2\pi h$, $h \in \mathbb{N}$ always holds for dual lattices. Hence for the non trivial \mathbf{G}_l , which actually backfold the original bands onto new bands $G_{l,I} (M^{-1})_{IJ}$ must be rationals. On the other hand $r_{j,J}$ are also integer but chosen such that they do not represent SC lattice vectors, since otherwise $r_{j,J} (M^{-1})_{IJ}$ must be integer for $\mathbf{G}\mathbf{R} = 2\pi H$, $H \in \mathbb{N}$. Altogether, when thinking hard one realizes that $\mathbf{G}_l \mathbf{r}_j$ runs over f fractionals such that the resulting f complex numbers $e^{i\mathbf{G}_l \mathbf{r}_j}$, $j \in [1, f]$ are equally spaced on the unit circle including the number 1 (for $\mathbf{r}_j = 0$). But then a general sum of unity theorem tells us $\sum_j e^{i\mathbf{G}_l \mathbf{r}_j} = f_s \delta_{\mathbf{G}_l \mathbf{g}}$, where any NC reciprocal vector \mathbf{g} will work. If we do partial backfolding, where $f_s < f$ the sum of unity is no longer correct, but one can argue that we get something with f_s terms of order one and an approximate delta function.

Hence, the average becomes (using Eq. (15.5))

$$\boxed{\frac{1}{f_s} \sum_j C_{[SC]S_{j_s\mu,n+lm}}^{\mathbf{k}} = \frac{1}{\sqrt{f}} \delta_{\mathbf{G}_l \mathbf{g}} C_{[NC]s\mu,n}^{\mathbf{k}} U^{\mathbf{k}}} \quad (15.7)$$

The unitary mixing only happens in degenerate subspaces and drops out of the final weight expressions if an average over the degenerate bands is taken.

In words: if the bands are from the non backfolded set, $l = 0$, $\mathbf{G}_l = \mathbf{g}$ we get $C_{[SC]} = \frac{1}{\sqrt{f}} C_{[NC]}$. If it is a backfolded set $l \neq 0$ and $\mathbf{G}_l \neq \mathbf{g}$ we get exactly (approximately) zero. Hence, the average defined above is shown to differentiate between backfolded and original bands. Of course one can continue this relation to \mathbf{k} -points outside of the first SC reciprocal unit cell and hence recover the full band structure of the larger NC unit cell.

What is left is the normalization condition. We will discuss net weights in order to stay consistent with the FPLO scheme of things. The sum over all standard SC weights must equal one. Using Eq. (15.6) we can write

$$\begin{aligned} 1 &= \sum_{S\mu} w_{S\mu}^{k,n+lm} = \frac{1}{\Omega} \sum_{S\mu} \left\langle \left| C_{[SC]S\mu,n+lm}^{\mathbf{k}} \right|^2 \right\rangle_{\text{deg } n} \\ &= \frac{1}{\Omega f} \sum_{S\mu} \left\langle \left| C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \right|^2 \right\rangle_{\text{deg } n} \\ &= \frac{1}{\Omega f} \sum_{j_s\mu} \left\langle \left| C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \right|^2 \right\rangle_{\text{deg } n} \\ &= \frac{1}{f\Omega} \sum_{s\mu} f_s \left\langle \left| C_{[NC]s\mu,n}^{\mathbf{k}+\mathbf{G}_l} \right|^2 \right\rangle_{\text{deg } n} \end{aligned}$$

The proposed unfolded weights using the r_j averages read (with an undetermined factor X_s)

$$\begin{aligned} w_{s\mu}^{kn+lm} &= X_s \left\langle \left| \frac{1}{f_s} \sum_j C_{[SC]S_{j_s\mu,n+lm}}^{\mathbf{k}} \right|^2 \right\rangle_{\text{deg } n} \\ &= X_s \frac{1}{f} \left\langle \left| C_{[NC]s\mu,n}^{\mathbf{k}} \right|^2 \right\rangle_{\text{deg } n} \delta_{\mathbf{G}_l \mathbf{g}} \end{aligned}$$

The normalization sum reads

$$1 = \sum_{s\mu} w_{s\mu}^{kn+lm} = \frac{1}{f} \sum_{s\mu} X_s \left\langle \left| C_{[NC]s\mu,n}^{\mathbf{k}} \right|^2 \right\rangle_{\text{deg } n} \delta_{\mathbf{G}_l \mathbf{g}}$$

Comparing to the sum of SC net weights above assuming that we have an original band ($\delta = 1$) we get $X_s = \frac{f_s}{\Omega}$ and the final normalized unfolded net weight result is

$$\boxed{w_{s\mu}^{kn} = \frac{\frac{1}{f_s} \left\langle \left| \sum_j C_{[SC]S_{j_s\mu,n}}^{\mathbf{k}} \right|^2 \right\rangle_{\text{deg } n}}{\sum_{S\mu} \left\langle \left| C_{[SC]S\mu,n}^{\mathbf{k}} \right|^2 \right\rangle_{\text{deg } n}}} \quad (15.8)$$

Finally, after having discussed everything for the exact supercell, we postulate that the same procedure is also applicable for slightly distorted supercells, which gives the unfolding procedure.

15.2.4 Summary:

Band unfolding is a fat-band method. By projecting the Bloch spectral density operator onto Bloch sums of the NC periodicity, the resulting weights will be strongest for the bands, which belong to the original NC band structure. The other bands of the SC, which are obtained by backfolding NC bands, will have smaller or zero weight, depending on the amount of perturbation, which differentiates the SC from a perfect duplication of NC cells. The philosophy of unfolding contradicts atom substitution. However, atom substitution can be handled too at least formally.

15.3 Perturbations

There are two kinds of perturbations, moving atoms and replacing atoms. Moving atoms means that the overlap matrix between the NC and the SC orbitals/WFs gets approximated in this unfolding technique and that the phase factors of the NC Bloch states are only correct on average, which usually is a small thing, if atoms are moved only slightly. However, one should keep in mind that the projections used assume perfect matching (which only exists in perfect multiples of NC). Moving physically unimportant atoms (no contribution to the energy window under consideration, e.g. Fermi level) is of course a potential-only perturbation for the important atoms (which contribute to the energy window) and hence unfolding for the important atoms makes sense.

Replacing atoms comes in two modes. Replacing unimportant atoms (charge donors, buffer atoms) really just changes the potential of the important (other) atoms and the unfolding for the important atoms is rather meaningful. Example: replacing cations in the pnictides while leaving the FeAs planes intact, gives meaningful unfolding for the Fe bands around the Fermi level. Replacing the important atoms gives problems for the following reason. Assume the NC and SC as shown in Figure 15.2a.

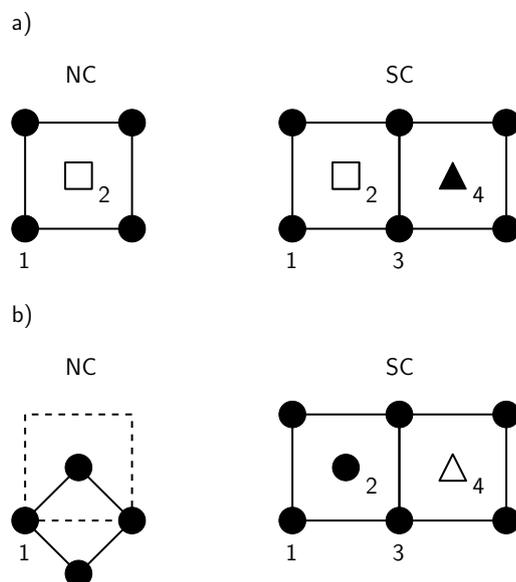


Figure 15.2: Example for replacing atoms

The sites $s = 1$ gets unfolded by equating sites $S = 1, 3$. This will lead to nicely unfolded bands. The unfolding of sites $S = 2, 4$ cannot be done in the technique used here, since the atoms and hence the basis of these atoms are different and there is no way of forming approximate Bloch sums of sites $S = 2$ and $S = 4$. Instead one has

to request unfolding with site list $S = 2$ and $S = 4$ separately. This just means that the unfolded band weights of these two sites are identical to the not-unfolded weights and hence show the SC periodicity (backfolding in the BZ). This makes some sense, since there really is no approximate NC symmetry for these atoms. The total unfolded weight (sum over all orbitals) for unfolded $S = 2$ is half as big as the total unfolded weight of $S = 1, 3$ due to the site count. In that sense the unfolding is still visible to a certain extent. In Figure 15.2b the situation is a bit better. Now, unfolded site weights can be defined from $S = 1, 2, 3$ and $S = 4$. The weights will now play out better, since only one out of four sites, which form a full NC Bloch sum is missing. So we can plot the $S = 1, 2, 3$ unfolded weights and the unfolded=not-unfolded $S = 4$ bands.

15.4 Brillouine zones

The NC bz is larger than the SC BZ. Example from Figure 15.2a: the bz is shown in Figure 15.3.

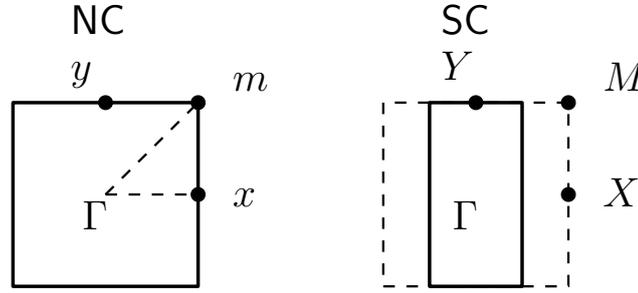


Figure 15.3: bz and BZ of Figure 15.2

In the NC we have the high symmetry points

$$\begin{aligned} x &= \left(\frac{\pi}{a_{[NC]}}, 0, 0 \right) = \frac{2\pi}{a_{[NC]}} \left(\frac{1}{2}, 0, 0 \right) \\ m &= \left(\frac{\pi}{a_{[NC]}}, \frac{\pi}{a_{[NC]}}, 0 \right) = \frac{2\pi}{a_{[NC]}} \left(\frac{1}{2}, \frac{1}{2}, 0 \right) \\ y &= \left(0, \frac{\pi}{a_{[NC]}}, 0 \right) = \frac{2\pi}{a_{[NC]}} \left(0, \frac{1}{2}, 0 \right) \end{aligned}$$

In the SC the equivalent points are at the same cartesian coordinates, but using $a_{[SC]} = 2a_{[NC]}$ we get

$$\begin{aligned} X &= \left(\frac{2\pi}{a_{[SC]}}, 0, 0 \right) = \frac{2\pi}{a_{[SC]}} (1, 0, 0) \\ M &= \left(\frac{2\pi}{a_{[SC]}}, \frac{2\pi}{a_{[SC]}}, 0 \right) = \frac{2\pi}{a_{[SC]}} (1, 1, 0) \\ y &= \left(0, \frac{2\pi}{a_{[SC]}}, 0 \right) = \frac{2\pi}{a_{[SC]}} (0, 1, 0) \end{aligned}$$

Although, only the x -direction changes in the new BZ, all fedit coordinates change. This is because the cartesian coordinates are the same in NC and SC, but the unit $\frac{2\pi}{a_{[SC]}}$ scales for all directions.

15.5 Caveats

- The unfolded weights might look much smaller than the weights in the NC along some directions. This can happen, when the bands are degenerate in the NC and is due to a splitting in the SC combined with

the way of counting. So, a width 1 band becomes, let's say, 2 width 0.5 bands shifted slightly against each other. Now, they appear to be half as broad, when the splitting is small and hence the bands are more or less plotted on top of each other.

- Take care, when determining the NC bz in the SC setup.
- Unfolded weights do not show the full information, because the Bloch spectral density is not diagonal in the approximate NC Bloch states.

15.6 User input/output

The user defines unfolding by creating the file `=.unfold`. If the file is detected the fatbands (`+bweights_unfolded/+bweights_kp_unfolded`) are created according to formula 15.8.

15.6.1 `=.unfold`

The file looks like this

```
# NCsite SCsites
1      1 2 3
# some comment
2      4
```

There can be any number of comment lines starting with '#', or lines only containing whitespace. They are ignored.

Each line, which is not a comment line and not whitespace-only is an unfolding definition.

An unfolding definition contains

the NC site number: which has no other use than labeling the contracted atoms in the label information in `+bweights_unfold`. Maybe it's best to use the actual site number in the NC.

the SC site list: which is the list of sites in the SC, which get contracted onto the NC site.

The number of SC sites, which are contracted to this NC site (in the formulas above it is f_s) is determined from the input. Usually it equals the number of times the NC fits into the SC $f_s = f$. In cases where atom substitutions took place $f_s < f$ as in the example file above: 4 NC cells form one SC, hence $f = 4$. Atom 4 got substituted and has a different atom than sites 1...3. So, we contract/unfold sites 1...3 and leave site 4 by itself. Note, that one can contract different atoms, as long as they have the same basis, i.e. in most VCA cases and when substituting similar elements (e.g. Fe with Ni).

15.6.2 *choosing k-points*

Beware of choosing the correct k -points. The unfolded fatbands are calculated in the SC BZ. So, first you determine the cartesian representation of the special points in the NC bz. Then you find the relation between bz and BZ and finally transform the bz points into BZ points and put them into the fedit menu in units of $\frac{2\pi}{a_{SC}}$. That means that a k -point in cartesian coordinates \vec{k} is entered as \tilde{k} in fedit with $\vec{k} = \frac{2\pi}{a}\tilde{k}$, where a is the first lattice constant.

15.6.3 Fermi surfaces

Fermi surfaces can be created with xfsf. Xfsf uses the symmetry to create an irreducible mesh and hence to save calculational time. The NC bz is larger than the SC BZ. Therefore, it is impossible to create enough NC k-points by using the SC BZ. There is a menu input->handmade symmetry, which allows the user to enter the NC cell and symmetry operations (only some generators are needed). This overwrites the default mesh symmetry. After this the process is straight forward, except for the file suffix, of the unfolded fatbands, which requires to set a non-default filename in input->files. Note that the new convention is to name the band file `+band_kp` for the Fermi surface (in general if `=.kp` was used), in order to not overwrite `+band`. So, the unfolded file is `+bweights_kp_unfold`.

The bandweights can be used for coloring: plot->coloring: extern, state. The extern checkbox switches from Fermi velocity coloring to extern coloring from the file given in input->files. State selects the data column from the file. (Transparent coloring as in Ref. [12] is not officially available, since it was a hard coded hack.)

15.6.4 Examples

There are some examples in the `FPLO22.00-62/DOC/Unfolding_example` directory. Use

```
fplo22.00-62-x86_64
```

to calculate `+bweights...`. Then use

```
faddwei22.00-62-x86_64
```

and if the file `=.addwei_unfold` exist (supercell sub directories)

```
faddwei22.00-62-x86_64 -s =.addwei_unfold
```

to extract `+bwsums...` and finally

```
xfbp bw.cmd
```

to show the results.



Chapter 16

Automation, scripting, pipe-mode, PYFPLO

The manipulation of input files via unix commands like `ed`, `sed`, `awk` and similar ones is strongly discouraged. The format of the input files follows a syntax and is not fixed in position. To achieve automation the pipe-mode of FEDIT can be used.

NEW: There is a new PYFPLO python package, which can be used for input file manipulation and which is documented here: [../pyfplo/pyfplo.pdf](#). It uses the mechanism described below for input creation, but is rather easy to handle. It also has the ability to read input files. It is more general.

16.1 Rules

There are only a few rules to be obeyed.

1. The menus/screens/edit-controls of FEDIT are operated by ascii hotkeys in interactive mode. For some actions there are control keys (cursor movement, searching and scrolling). In pipe mode the latter keys are not needed at all. The ascii hotkeys are replaced by a simple syntax explained below.
2. In interactive mode sometimes only a portion of the menu's form is seen on screen. In pipe-mode always the full form is virtually visible. There is no need for scrolling. (In fact there is no need for scrolling in interactive mode as well, since the currently invisible hotkeys are also active and just typing an currently invisible hotkey will scroll the form to make the selected edit-control visible.)
3. In interactive mode the edit-controls may be edited to change only part of the controls content. In pipe-mode the full content/data of the edit-control has to be entered.
4. In interactive mode there are toggling actions, in pipe mode these become normal edit-controls.
5. In interactive mode there will be informational screens displayed after certain actions, which just show output. In pipe-mode these screens will not occur. (E.g. the FPLO message after symmetry update.)
6. In interactive mode there will be questiones to be answered depending on the context. In pipe-mode such questions naturally cannot occur.
7. In interactive mode, the user enters information by typing keys, in pipe-mode the user input to FEDIT is read from standard input (`stdin`). This input may e.g. be stored in a file or come from a `here-script` within a shell/perl-script.

8. In interactive mode the user feedback is what is seen on the screens, in pipe-mode the feedback is written to standard error (`stderr`). So it is good practice to redirect `stderr` to a file and to redirect standard output to `/dev/null`. (The screens on `stdout` will change so rapidly that they are useless.) Let us assume that the input for the pipe-mode is stored in the file `=.pipe`. (The prefix indicates that it is an essential input file.) A good way to feed the information contained in `=.pipe` into FEDIT in a shell environment would be

```
cat ./=.pipe | FEDIT -pipe 2>./+log 1>/dev/null
```

or

```
FEDIT -pipe < ./=.pipe 2>./+log 1>/dev/null
```

Both commands have the same effect.

- (a) The first command uses `cat` to write the content of `=.pipe` to `stdout`, which than is redirected to the `stdin` of FEDIT via the unix pipe command `|`. (That is the origin of the name pipe-mode.) The second command uses the unix tool for redirecting the `stdin`. Here the content of `=.pipe` is directly written to the `stdin` of FEDIT.
 - (b) The editor is given the option `-pipe` to setup the mode.
 - (c) The `stderr` is redirected to the file `./+log`. (The naming is up to the user, however, our choice follows the rules explained in Chapter 3. The '+' indicates that the file is not essential and may be deleted after a succesful run.)
 - (d) The `stdout` is redirected to the unix device `/dev/null`, which just means to discard it. (`/dev/null` is a 100% information sink :-)
9. The return code of FEDIT should be checked and the script aborted if needed, to intercept input errors. FEDIT sets the shell exit/return code as

```
1      on success
```

```
other  on error
```

The exit/return code must be checked immediately after the command, which returned it. Any shell action in between changes the exit/return code!

10. The logic of the pipe-mode is such that the user has to feed hotkey-data sequences and menu-hotkey sequences into FEDIT as he would in interactive mode. The user creates key sequences in the pipe-input, which navigate through the menus as if it would be an interactive session. This includes the `x`-hotkey to leave a sub menu. It excludes the `x`-hotkeys of information screens as described above. (The main reason is, that those screens are context dependend.) If there is any invalid input in the pipe file, the editor will abort unsuccessfully. The reason for the failure is printed to `stderr`, which is accesible as explained above. To create pipe input, just use the editor interactively and write down or remember all hotkeys, which are pressed to complete the desired editing (except the information screens and questions).

Altogether the simple rule is that in pipe-mode there is only navigation to certain positions and entering of data. Every other user \Leftrightarrow FEDIT interaction will be absent.

16.2 Syntax

We assume that the pipe input is stored in a file. Its syntax is as follows.

1. The file may contain comments, which are lines whose first non-blank character is '#'.
2. The file may contain empty/blank lines.
3. Every single action goes into a separate line.
4. A hotkey is wrapped into two '@' characters. (If `perl` is used for the input creation the character '@' has a special meaning in `perl` and must be protected '\@'.)
5. An alternate sub menu ('<SPACE>hotkey') is called with a single space followed by the hotkey, wrapped into two '@' characters

```
@ hotkey@
```

6. A hotkey sequence, which activates an edit-control to enter data is mapped onto a line

```
@hotkey@data
```

7. A hotkey sequence, which activates N edit-controls is mapped onto

```
@hotkey@data1@data2@...@dataN
```

Example: The symmetry menu contains the Wyckoff position definitions, which consist of an element name and a vector. The hotkey equals the sort number. A corresponding pipe input to edit the second sort could look like

```
@2@ Co @ 2/3 1/4 ,
```

This sets the Wyckoff position definition for sort 2. The comma ',' is a duplicator of the value preceding it¹. Thus

```
1/4 , 2/3 = 1/4 1/4 2/3
1/4 ,, = 1/4 , , = 1/4 1/4 1/4
```

8. An edit-control may be activated by a search command (as in interactive mode). This is done by

```
!search-string!data
```

This will look for the first occurrence of the search string in the current form and if this marks an editable control, it is activated and data is used as the control's new data. Multiple edit-controls, selected by search are mapped onto

```
!search-string!data1@data2@...@dataN
```

(Note: As in the previous point, the '@' replaces the '<ENTER>' key used in interactive mode.) As in interactive mode, the parentheses enclosing the hotkeys on the FEDIT-screens are ignored in search mode. Example: the screen entry "Conver(G)ence condit" is searched for as "convergence condit".

9. An edit-control, which is a toggle in interactive mode is not toggling in pipe mode. So, if such data shall be edited in pipe mode one needs to set the value explicitly. This assures a defined state of the input after completion.

- (a) **Old:** logical values may be 't' or 'f'. **Now,** the values are no longer differently represented on screen compared to options and can be set by 't', 'f' or '+', '-'.
- (b) binary values take their values as they are written on screen in interactive mode. (Example: spin sorts may be '1' or '2')
- (c) Options (marked as selected by [X] on screen) are selected or deselected with the values '+' or '-' in pipe mode

Example:

To select the option `NO_SYMMETRYTEST` in the options submenu we may use the search tool

¹The comma also works in interactive mode.

```
!NO_SYMMETRYTEST!+  
or to deselect it use  
!NO_SYMMETRYTEST!-
```

The search string must be chosen to be unique in the way that it really selects what you want. It is best used only in options menus and in select boxes.

It is strongly discouraged to activate options (the things marked with '[X]') via the related hotkey, since these hotkeys are created automatically by FEDIT and thus may change if the version changes². The search utility is designed exactly for the purpose of changing options. In select boxes the hotkeys will not change. However, the search utility gives better readable pipe-files.

²In fact, there is a list of all possible options. FEDIT takes this list and creates hotkeys for all of them. If the order of options in this list changes between different FPLO versions, the mapping of hotkeys to options will naturally change as well. The authors try to avoid a change of the options order. However, there are thinkable cases when this will be necessary!

10. The alternative choices in a select box are selected by a single hotkey command as in

```

...
# we assume we are in the main menu here
# we enter the relativistic select box (hotkey 'r' in the main menu)
@r@
# we select the full relativistic mode, by searching for
# 'full relat', which is unique in this select box
!full relat!
# We equally could use the hotkey, off course.
# Let us select it again:
@f@
# Now we leave the select box and go back to the main menu
@x@
# continue
....

```

Please observe that the parentheses indicating the hotkey ('(F)ull relativistic' in this case) are not visible in search mode, interactive or not.

11. The last key sequence in the pipe mode must be

```
@q@
```

and must be called being in the main menu. (So, you need to virtually navigate back to the main menu.) This is to assure proper cleanup.

16.3 How to set up automation

There is one peculiarity. Suppose you created a series of input files and converged all related calculations. Now, you want to change some settings, for instance increase the number of k -points, since it turned out that you had too few of them. You could do this with a minimal script, which only changes the number of subdivisions in k -space. Later you recognize that you need to change other settings. Again you could do it with a minimal script. The main drawback of this approach is, that you lose control over the changes, which have been done. Another point is that after a change of certain symmetry parameters some input data are reset to default values. For these reasons it is advisable, to set up the pipe file(s) always such that the input files are completely determined by the content of the pipe file.

If FEDIT is called within a directory containing input files it offers the content of these files for editing. This means that the state of input files depends on the previous editing. The following sequence of actions will assure a state that does not depend on previous editing.

1. Go to the symmetry menu and enter all symmetry parameters explicitly. (These are not so many data, so it is not a big task.)
2. Call `update` with the '+' hotkey! This will reset certain data of the `=.in` (and `=.basis`) files. **Never forget this point!** Most of the settings will remain unaltered. All calculations of a series should belong to the same symmetry type. That means, the Wyckoff positions should at most change their parameters but the spacegroup should stay the same.
3. Go back to the main menu (hotkey 'x')
4. Call the alternate submenu action `Recreate` (hotkey '<SPACE>e'). This will reset the file `=.in` to default values, which belong to the current symmetry setup. Remember that in pipe mode the corresponding

question (Really rebuild default content ? (y/n):) will not be asked, so you must not type 'y' or 'n' in pipe mode. Following the answer 'y' an informational screen will appear in interactive mode, which will be left with 'x'. This screen will be absent in pipe mode, due to the general rules above. That means that in interactive mode you would type the sequence '<SPACE>eyx' to perform the rebuild, but in pipe mode the action is achieved with a single line

```
# assume we are in main menu
# next line will call the reset action
@ e@
# we are back in main menu now, having default settings
```

5. Now, enter all input data, which differ from default input, except symmetry data.

A setup for a series of calculations may be done as follows.

1. Create interactively an initial input for your compound. Perform a self consistent calculation. Check if everything is fine. Converge the number of k -points (may already be done with the help of scripts, off course).
2. Set up the scripts, which create the input for the series. Make sure to perform every calculation in a separate directory! After creating a new directory for a particular calculation of the series and before invoking FEDIT copy `=.dens` (`=.basis` for older FPLO versions) from the initial calculation directory into the new directory. This has the advantage that the pre-converged density will be available. As a result convergence will be much faster (in many cases) than in a calculation from scratch. This procedure makes sense as long as the parameter variation in the series is not too large. But even if so, the proposed procedure will not hurt.
(In calculation series, (e.g. search of the minimum of an energy surface using e.g. a steepest decent algorithm) where the input parameters depend on previous calculations, one may copy one of the latest previous calculations into the new directory instead using the initial calculation.)
Make sure that the `'=. '`-files are copied only, if a new directory has been created. (Do not overwrite converged basis or density.)³ If the script is re-run with some parameters changed, the density of the previous self consistent calculation should be retained!
If other input files are used (e.g. `=.basdef`) make sure that they are copied too or that they are created by the script.
3. Run the script to create the directories with the proper input. Check the created input interactively, especially when new scripts are developed!
4. Launch all calculations in the various directories and wait for self consistency. Redirect output into a file, best is to always use the same file name (for instance 'out'). Check from time to time if everything runs fine.
You may modify your script such that it serves the input creation and the starting of the calculations. Or you create a separate script to launch the calculations. Use the specifics of your platform (may be there is a job queue).
5. Check if all calculations really converged. There is a final message on completion of a calculation⁴. Check this. The final message has the structure

```
TERMINATION: Keyword : explanation
```

where keyword may be

³This means that the script, which creates the directories and the input should test the existence of the directory.

⁴Except a bug occurred.

Finished	The calculation run to self consistence or a single step calculation finished.
Normal	The calculation did what it was ment to do and achieved the goal. Some actions terminate the program before self consistency, and this triggers a normal termination.
Error	Something happend. In many cases there is a cure for the problem.
Crash	This is really a bad case.

The Keyword may be checked automatically with the help of unix tools. The explanation string gives a rough idea, what happend.

6. Have a look at example scripts in the distribution on how to extract certain data from the output.
7. Now, you are ready and may modify the script to change some settings and re-run the series if nessecary. Take care of all points mentioned so far.

16.4 Advanced features

16.4.1 Initial polarization, inital spin split

Some times, you may encounter a situation where it is nessecary to repeat an initial spin split for a series of calculations, since for example, the calculations were started spin polarized with an insufficient split and run back to a non magnetic state. Instead of discarding all calculations done so far, it is better to force a new inital split on the pre-converged density. But, we face the situation that there are already 2 spin sorts in the density files and thus another `inital polarization` would urge FPLO to pose the question

Do you want to skip the possibly repeated initial spin splitting ?

Normally, the answer would be 'y' since mostly one just forgot to switch off the `inital polarization`. But in our situation the answer would be 'n', which forces another split. (What we really want is to restart the calculations in a new attractor basin.)

Now the problem is that usually the FPLO jobs are started in a background mode. In such a case there is no user to answer the question and so the job will crash (due to the fact that it tries to read from `stdin`, which is not present in background mode). If the job is not running in background it will hang and wait for the answer. Imagine, you started the script before weekend and come back at Monday and it still is waiting for the answer, or crashed.

There is a way to circumvent the problem: One provides FPLO with the nessecary input on `stdin`. If the program is not reading from `stdin` (since the question did not arise) the `stdin` will just be ignored and no problems occur. But if the question arises the proper answer is available and the program continues. There is an easy way to do this.

In the script which actually lauches the job there will be a line like

```
FPLO 2>/dev/null >out
```

We have to change it into

```
FPLO < ./+yes 2>/dev/null >out
```

or

```
cat ./+yes | FPLO 2>/dev/null >out
```

where the file `+yes` has to be created before (by the script) in the directory, where FPLO is running. This file contains a single line with the proper answer ('y' to skip the split in almost all cases or 'n' in the particular situation, which we discussed above.). The file may be created with a `here-script` like in

```
cat <<EOF > ./+yes
n
EOF
```

or with `echo` like in

```
echo "y" > ./+yes
```

So, coming back to our situation. To enforce the repeated spin split we would (in the script) switch on the `initial polarization` and we would put an 'n' into the file `+yes`. After a successful split (means after the re-started calculations passed the splitting), we should immediately replace the 'n' by an 'y' and switch off the `initial polarization` in the script, to avoid an unwanted destruction of the now hopefully correctly converged results.

In fact one should always use this construct, just in case, that one forgot to switch off the `initial polarization`. Some people will know the unix command `yes`, which will do a similar job as the file construct. However, in practical applications we often met the situation, where this command was not available in a job-queue environment. Therefore, we decided to use the file trick.

16.5 Example

We give an example script here, which uses basic features explained above. It is contained in the distribution directory (in case of default installation something like `FPLO/FPLO22.00-62`) under `DOC/Scripting_example/`.

```
#!/bin/sh
#
# Example script to create a series of calculations for varying lattice
# constant for fcc Al.
#
# We use the so called here-script mechanism to create the pipe input
# on the fly.
# This script works with bash at least. For other shells part of the syntax
# may be different. Consult your man pages.
#
# We assume that the script is executed in a directory, where there is
# an initial calculation sub directory called SC containing a converged
# calculation of the same compound (fcc Al).
#
#####
```

```

# Always use fully qualified names, to assure proper program version.
# Store the exec names in variables, so we do not need to scan the script
# to replace the version later on.
FEDIT=fedit22.00-62-x86_64
FPLO=fplo22.00-62-x86_64

# Give all directories a name prefixed with the name of the parameter,
# which is running, followed by the parameter itself
prefix='a0='

# Remember the current directory.
# Be aware of the back quote syntax, on some unix systems you need a different
# construct to cast the output of a command (pwd here) to a string.
ROOT='pwd'

#####
# some functions
usage() {
  echo "usage: $1 [-r] [-h[elp]]"
}

#####

# Check the command line flags.
#
RUN_IT=0

while :
do case $1 in
  -r) RUN_IT=1
      shift 1
      continue
      ;;
  -h*)
      usage 'basename $0'
      exit
      ;;
  -*) usage 'basename $0'
      echo 'wrong options'
      exit
      ;;
  *) break
      ;;
esac
done

```

```

#
# security questions
#
if [ x$RUN_IT = x1 ]
then
    echo "Shall I run the jobs: [y/n]" ; read YN
    # The next test is a little trick to circumvent problems with
    # empty variable. We first form the concatenation x$YN, which
    # has the value of $YN prefixed with a single x.
    # Then we compare it with the teststring (y in our case) prefixed by x.
    # So, if $YN=y than also x$YN=xy. Disturbing?
    if [ "x$YN" != "xy" ] ; then
        echo "abort"
        exit
    else
        echo "Will run jobs now."
    fi
else
    echo "Shall I (re)create the input: [y/n]" ; read YN
    if [ "x$YN" != "xy" ] ; then
        echo "abort"
        exit
    else
        echo "Will (re)create input now."
    fi
fi

#####

# loop over the running parameter, in our case the lattice constant
for xx in 6.50 7.00 7.50 8.00 8.50
do

    # make sure we are in the root directory of our data directory tree
    cd $ROOT

    # create the directory name as described above (example 'a0=6.00')
    dir="$prefix$xx"

    # check, if input creation or job running shall take place
    if [ x$RUN_IT = x0 ] ; then

```

```

# input creation branch

# if the directory does not yet exist, create one
if [ ! -d $dir ]
then
    mkdir $dir
    echo "directory $dir created"
    # copy the essential files from the initial calculation
    # which is assumed to be in the directory SC
    cp ./SC/= .dens $dir
    echo " =. files copied"
else
    echo "directory $dir exists already"
fi

# change into the directory of parameter $xx
cd $dir

# Now create the pipe file content, with the help of a here-script.
# For the sake of book keeping we will save the pipe info into a file.
# (One could equally pipe directly into fedit.)

# The next command is the here-script ( <<EOF ), whose stdout is
# redirected ( > ) into the file ./=.pipe.
# Everything which comes between the <<EOF line and the line below,
# starting with EOF, will go into ./=.pipe. The main advantage of this
# approach is, that we may use shell variable replacement
# (interpolation) to put the information of the running variable
# $xx at the proper position
cat <<EOF > ./=.pipe
#####
# this is the beginning of the pipe file

# go to symmetry menu
@+@
# title
@c@A1, a0-variation
# enter spacegroup select box
@s@
# select space group
@225@
# leave selectbox
@x@
# structure type

```

```

@t@
  # crystal
  @c@
  # leave select box
  @x@
# lenth units
@u@
  # bohr radii
  @b@
  # leave select box
  @x@
# lattice constants; Here we put our running variable.
# We enter as first lattice constant the value which is in $xx,
# then we use the fedit-','-syntax to repeat the value
@l@ $xx , ,
# set axis angles
@a@90.,,
# setup Wyckoff positions
# number is one in our case
@n@1
# Now, give list of ALL !!! Wyckoff positions.
@l@ Al @ 0.,,
#
# NOW CALL UPDATE, NEVER FORGET THIS!!!
#
@+@
# leave symmetry menu
@x@
# back in main menu
# This was the symmetry setup, and now we follow our advise to create
# the default =.in input by using the REBUILD-action.
# (The space before the 'e' opens the alternative menu bar.)
@ e@
# now we have the default input, and are still in the main menu

# Let us set the number of k-points to a non default value:
@k@ 16,,

# Let us set the xc-potential version now:
# first enter select box
@v@
  # select via search
  !Perdew Wang 92!
  # leave select box, go back to main menu
  @x@

#Let us set the relativistic mode:
@r@

```

```

# Select by search, please note that the parentheses indicating the hotkey
# are not considered in search mode.
# (Have a look at the select box interactively.)
!scalar relativistic!
# leave select box
@x@

# Let us set some options:
# enter options menu
@-@
# Good habit is to select all options explicitly
# This includes the options, which are selected by default.

# here an example for deselecting
!PLOT_BASIS!-
# here an example for selecting,
!NO_SYMMETRYTEST!+

#leave menu
@x@

# Let us select spin=1 explicitly:
@s@1

# Let us switch off initial polarization explicitly:
@i@f

# last action must be
@q@

# this is the end of the pipe file
#####
EOF

# Now execute fedit in pipe mode and use the information of the file
# =.pipe just created in $dir.
# We made sure that we are in $dir, since fedit shall act there!
# We no longer explicitly tell fedit which fplo executable to use!

```

```

$FEDIT -pipe <./=.pipe 2>./+log 1>/dev/null

# Check exit/return code of fedit, there must not be any command
# in between the check and the command, which produced it (fedit here).
# The return code is stored in the variable $? in shell.
# exit=1 means success
if [ $? -ne 1 ]
then
    cat<<EOF
Content of log file:
-----
EOF

    cat ./+log

    cat <<EOF
-----
There was an error in the pipe input. Check logfile above or in $dir/+log.
Be aware that the line numbers refere to the file $dir/=pipe!
EOF

    exit 2;
fi

# If we are here, the input was created in $dir according to our setup
# Now we continue with the next parameter

# change back to where we started
cd $ROOT

# end of input branch

else

# job-run branch

# change into the directory of paramter $xx
cd $dir

echo "$FPLO running in $dir ..."

# now execute, whatever is nessecary to launch job in the current
# directory (name $dir)

#START: example
# We just run the jobs sequentially on a single machine
# and redirect stdout to file out and stderr to /dev/null.
# (In this way there will be no dangling output and the job could run

```

```
# savely in background, which is not done in our example here.)

# Furthermore, we use the +yes-file mechanism to avoid a crash
# due to repeated initial polarization (spin split).
# The "y" below enforces fplo to continue in such situation
# without a repeated split and does nothing otherwise. See manual.

echo "y" > ./+yes

cat +yes | $FPLO 2>/dev/null > out

#END: example

# change back to where we started
cd $ROOT
fi

# end of xx-loop
done

if [ x$RUN_IT = x1 ] ; then
    grepfplo -p 'a0=' -m EE | tee e
fi

#
# After the input creation run we should have a directory structure like
#
# ./SC/
# ./a0=6.00/
# ./a0=6.50/
# ./a0=7.00/
# ./a0=7.50/
# ./a0=8.00/
# ./a0=8.50/
# ./script
#
# where every directory contains the same setup, except for the
# lattice constant.
# We may now perform converged calculations (option -r) in all directories.
# If we want to change, say, the number of k-points, we edit this number
# in the pipe-section above, re-run that script to change the input and
# re-converge the calculations (option -r).
#
```

To test the script, copy the directory `Scripting_example` (including the subdirectory `SC`) to an appropriate place, change into the copy and edit the variables `FPL0` and `FEDIT` in `fscript` to point to the fully qualified executable names of your installation. Make sure that the `PATH` is properly set, so that the executables may be called in your environment.

Then execute

```
./fscript
```

and answer 'y'. The directory structure described at the end of the script should have been created now. Next execute

```
./fscript -r
```

and answer 'y'. Now the calculation is running in sequence. Wait for it to finish. Test convergence via

```
grepfplo -p a0= -m it
```

Check termination status via

```
grepfplo -p a0= -m term
```

Collect total energies in file named 'e' via (done in the script with option `-r`)

```
grepfplo -p a0= -m EE | tee e
```

Plot file 'e' with e.g. XFBP if you want to.

The old scripts '`gr.`' are part of the installation and should be available if your installation procedure succeeded. But now you can use GREPFPLO instead.



Appendix

A Summary of Changes

FPLO22, Release 62: 1. General:

- (a) A bug in the optics module was found and fixed, which leads to too few tensor components ($\text{Im}\varepsilon_{ij}(\omega)$) being written to the file `+imeps`. This bug affected monoclinic and triclinic lattices, in which case the error was quite obvious.
- (b) For broadening BZ-integration methods the free energy and the extrapolated energy are additionally printed to the output now. See Sec. 5.3.
- (c) The (modified) Becke-Johnson xc-potential has been implemented, Sec. 5.5.1.
- (d) An option to adjust the accuracy of the numerical three-center-integrals was added to the FEDIT Numerics submenu (see Sec. 5.4).
- (e) The total gap, the mBJ-parameters (g_{mBJ} , c_{mBJ}) and the broadening-corrected and free energy are grep-able via `grepfplo`. Sec. 5.3 and 7.

2. Input:

- (a) The basis can be extended/modified in the FEDIT menu `Basis`. See Chapter 4. The file `=.basdef` is no longer the standard way of extending the basis. For non-standard modifications `=.basdef` can be manipulated via `pyfplo.fploio.Basis` ([./pyfplo/pyfplo.pdf](#)).
- (b) A new FEDIT submenu `XC-options` was added for setting adjustable parameters of xc-functionals. The mBJ-like potentials (Sec. 5.5.1) have parameter c_{mBJ} , which is determined self-consistently. For pre-converging calculations or for other reasons this parameter can be set constant in this submenu.

3. PYFPLO (see [./pyfplo/pyfplo.pdf](#))

- (a) We added `pyfplo.fploio.Basis` and related classes to manipulate the basis on low level
- (b) We added `pyfplo.fploio.OutGrep` to grep FPLO output from scripts, including the list of sites.
- (c) We added the list of elements: `pyfplo.common.c_elements` and `pyfplo.slabify.c_elements`.
- (d) We added `pyfplo.fedit.basis` related to the corresponding new FEDIT submenu.
- (e) We added `pyfplo.fedit.xcoptions` related to the corresponding new FEDIT submenu.
- (f) We added the THCI-settings to `pyfplo.fedit.numerics` related to the corresponding new FEDIT Numerics submenu settings.
- (g) Several PYFPLO examples were added.

FPLO19, Release 60: 1. General

- (a) A bug in the creation of high symmetry points has been fixed. This affects simple trigonal systems, which unfortunately were showing the high symmetry points of simple tetragonal.

2. Input:
 - (a) In Z_2 mode the symmetry operator eigenfunctions are written to output.
 - (b) A new k-mesh creation routine has optionally been added. It allows to pick isotropic subdivisions for centered lattices (of course still with symmetry restrictions, where they apply, e.g. in bct: $N_a=N_b$ but now $N_c \neq N_a$ is possible) The option is in the main FEDIT menu, hotkey M. See Sec. 5.1.
 - (c) In both k-mesh creation routines a subdivision of 0 triggers automatic subdivision. See main FEDIT menu help screen for explanation. See Sec. 5.1.
3. XFPLO
 - (a) The `export` button in the symmetry menu of XFPLO has now a cif-export option.
 - (b) The symmetry operations can be visualized in structure mode.
 - (c) The high symmetry points in the Fermi-surface module are classified according to symmetry. Points, lines and optionally planes and general points are available.
 - (d) A multitude of zooming, moving and rotation features have been added in the View menu.
4. PYFPLO
 - (a) PYFPLO can now be create as a python3 package. In FPLO/PYTHON execute:


```
make python3
```

 to make it.
 - (b) The Berry curvature in PYFPLO is now corrected for lack of symemtry. This is achieved by adding an approximation of the position operator matrix elements. The result for the periodic gauge has proper symmetry and does not change topological properties. `hamAtKPoint` can be forced to return the relative gauge (`gauge='forcereative'`) if `makedhk==True`. The PYFPLO interface for Wannier centers (TI) has now an option to use the relative gauge with proper correction terms, which yields properly symmetric Wannier center curves. The topological indices should not be affected.
 - (c) The whole `.-`file content can be scanned into a json file. See [../pyfplo/Examples/fploio/equaldot2json](#).
 - (d) Mirror Chern numbers are implemented. See [../pyfplo/pyfplo.pdf](#).
 - (e) The symmetry operators of Wannier function Bloch sums and operators are accesible. See [../pyfplo/pyfplo.pdf](#) and the example script [../pyfplo/Examples/slabify/symmetryops](#), which checks symmetries and demonstrates usage.
 - (f) The PYFPLO module `FPLOInput` has a cif-file import option with optional symmetry determination function called `structureFromCIFFile`. See [../pyfplo/pyfplo.pdf](#).
5. Wannier function module
 - (a) There is an `automode` option in the `wanniertools` to create automatically all Wannier functions for the valence sector or for all semi-core and valence orbitals. See [../pyfplo/pyfplo.pdf](#) and Sec. 6.2.2.
 - (b) The xc-field (option `makexcfield` in `hamAtKPoint` and `savebfield` in `WanDefCreator`) can be obtained as a separate operator in `+hamdata` and hence PYFPLO. See [../pyfplo/pyfplo.pdf](#) and Sec. 6.2.2.
 - (c) The position operator (basis connection and curvature) (option `makebasisconnection` in `hamAtKPoint` and option `savepositionoperator` in `WanDefCreator`) can be obtained as a separate operator in `+hamdata` and hence PYFPLO. See [../pyfplo/pyfplo.pdf](#) and Sec. 6.2.2. Example: [../pyfplo/Examples/slabify/Fe/SP/slabify/AHC/README](#).

FPLO18: 1. Input:

- (a) The file `=.sym` has been removed. All functionality including symmetry updates are based on `=.in` now.
- (b) FEDIT is now tightly bound to the FPLO version. This means that it is hard linked against the input management libraries of the corresponding version. The option `-p` has been removed.
- (c) The new symmetry treatment is more general. Old projects and new projects may not be compatible. Especially site coordinates might be different (although equivalent), which matters for `difvecs` in `=.wundef` and `=.bwdef`.

This should not affect completely new projects.

Topological invariants for inversion symmetric systems might look different: we print the invariants after each Kramers pair. The new symmetry module can contain trivial shifts in the non-symmorphic translations (compared to the old symmetry module). Such shifts can lead to sign changes of the δ_i at the TRIM points. If the changes take place inside a fourfold degenerate manifold with mixed parities (for some non-symmorphic space group irreps on the zone boundary) this sign change is not applied in the old order. Hence, the printed invariants change. However, in this case they do not have a meaning (no gap). So, no harm done.

2. Wannier function module:

- (a) spin-mixed relativistic Wannier functions are possible
- (b) individual local spin axes can be defined for each projector (see Sec. 6.2.2)
- (c) real space representation of WFs can be loaded into XFPLO (see below).
- (d) new output file `+hamdata` for use in `pyfplo.slabify` (see below).

3. dHvA module added. See Sec. 13.

4. Z2 invariants for non-centro symmetric topological insulators added. See examples section in [../pyfplo/pyfplo.pdf](#).5. Python module to manipulate input added (useful for scripting). [../pyfplo/pyfplo.pdf](#)

6. Slabify added. This is a python interface, which allows to map a Wannier function model (or hand written tight binding model) onto an ideal finite or semi infinite slab (or other less important structures). It can calculate the band structure (finite slab) or surface spectral density (semi infinite). It also has options for Weyl semi metals and topological invariants. See Sec. 14.

7. XFBP

- (a) python bindings for scripting of XFBP. (The old native scripting is still there.) See Sec. 10.
- (b) a dens-plot mode which plots things like spectral densities (pixelized data).

8. XFPLO

- (a) More space group settings.
- (b) A global cell rotation to keep a certain cell orientation after cell transformations.
- (c) Symmetry/cell manipulation tools.
- (d) A cif-file importer.
- (e) Atom labels via the `Plot>Labels` dialog.
- (f) It can now load Wannier functions and grid-plot data. For this purpose the `wfdata...` files have a header now. Similarly, the grid-output files `grid_...` have a different name (no leading plus) and a header. The `opendx` files are now only created when special options are set. See e.g. [??](#), on page [29](#) and on page [29](#).

- FPLO14:**
1. Internally a correction for diverging non-spherical density parts in scalar relativistic calculations was added, which leads to slightly changed total energies as compared to previous FPLO versions.
 2. Multiple compilation branches possible (`install/Makefile 1`). See Sec. 17.
 3. New tools, see Sec. 14.
 4. The basis can be changed. 21
 5. Density mapping: Sec. 9.4
 6. New input files, some changes of old input files into FEDIT menus: Sec. 3
 7. Band unfolding: Sec. ??, Chap 15
 8. Molecular/Individual band weights: Sec. 9.3
 9. Optics: Sec. 12
 10. Output of energy and band resolved real space densities/wave functions for plotting. (See FEDIT GRID OUTPUT submenu)
 11. Z_2 invariant for topological insulators with inversion centers. This can be found in an FEDIT submenu.
 12. Reduced exchange field (LxSDA, GGax). This can be found in the FEDIT menu.
 13. Fixed-spin-moment works now in full-relativistic mode. Note, that this is not as stable as in non-full-relativistic settings due to the fact that spin is not a good quantum number in full-relativistic mode, which requires an iterative scheme for full-relativistic FSM.
 14. The charging has enhanced options. Now, one can use Virtual Crystal Approximation (VCA), molecular ionicities (for molecules) and a jellium model for 3D-solids. The old files `=.atcharge` and `=.mol_charge` have been removed and placed into an FEDIT submenu. Data of these files are transferred to `=.in` on version update.
 15. The production of `bravais.ps` and `primitive.ps` is discontinued, due to the availability of XFPLO.
 16. The file `=.coeff` has been turned into the switch `Output +coeff file` in the FEDIT bandstructure submenu.
 17. Full relativistic LSDA+U (but still very bad convergence)
 18. Note, that we changed the table for the onsite orbital momentum to show entries for each site not each sort. This is repetitive information, but more consistent with other tables.
- 2009:**
1. Geometry optimization of internal parameters (Wyckoff positions) via forces.
 2. Flexible grid output of density, spin-density and potential for visualization. (Now, in an FEDIT sub-menu and with `opendx` output.
 3. Increased performance for larger cells and heavier atoms.
- 2008:**
1. Generalized Gradient Approximation (GGA).
 2. Scalar relativistic version in Koelling Harmon style
 3. Finite nucleus has been implemented for upcoming use with EFG calculations.
 4. Re-implementation of VCA.
 5. Test version of Wannier functions implemented.
- 2007:**
1. Currently the basis is predefined. The user is not expected to have to change the basis.
 2. The basis is fixed, therefore the old basis optimization is discontinued.

3. New potential construction and local orbital integration with higher accuracy.
 4. Molecule/Cluster extension.
 5. Major rewrite of the code.
 6. Forces, but not geometry optimization module yet
- 01.07.2005:**
1. file `=.ldos` is obsolete now, added to FEDIT menu
 2. automatic Ewald parameter and Fourier component choice implemented
 3. Orbital moment output for full-relativistic calculations
 4. **Experimental:** Core-confinement for $4f$ -systems
 5. minor changes in atom potential to increase stability.
- 01.07.2004:**
1. Full relativistic mode implemented (not for CPA and LSDA+U).
 2. New internal mesh settings for increased accuracy.
 3. Accuracy improvements in various places.
 4. CPA and LSDA+U work together (not full relativistic).
 5. Basis orbital grouping implemented for enhanced stability and avoiding of fixed x_0 .
 6. Large-scale code restructuring in symmetry treatment in order to implement the full relativistic mode.
 7. Format of `=.dens` changed.
- 03.03.2003:**
1. New shape function implemented, which should make voronoi cell merging obsolete. This has an influence on the total energy within the mHartree range.
 2. New spherically averaged crystal potential (orbital calculation) introduced, which now is the default. This changes the basis definition and therefore the values of the optimum x_0 . It results in a change of total energies. For 3d metals the new basis is a bit worse than the old. However, inclusion of 4d polarization states will, as before, converge the basis to a high degree. For more open structures like oxides the new basis scheme is more stable and in many cases better.
 3. CPA implemented.
 4. LSDA+U implemented.
- 31. Oct. 2002:**
1. A single step calculation (`niter=1`) will not overwrite the `=.dens` and `=.basis` files
 2. A bug in the `symmetrytest` is fixed.
 3. Enhanced accuracy in the two-center integrals.
- 10. Jun 2002:**
1. Change in scalar relativistic definition. (Tiny total energy differences < 1 mHa expected.)
 2. Kinetic energy correction due to finite radius of orbitals added. Changed results expected, if the basis contains orbitals with large non-vanishing derivative at the outermost mesh point.
- 06. May 2002:**
1. Bugfix for COMPAQ-fort Linux-alpha compiler in `bzone.f90`.
- 12. Mar 2002:**
1. Definition of empty sites changed: The addition of empty sites to a normal structure will no more change the basis definition of the normal atoms. Thus, the addition of empty sites has to lower the total energy (which was not the case before)!
 2. Total crystal density may be output using a grid definition file `=.densgrid`. See Section 3.

B FAQ

Q: How can one use existing input files with different versions of FPLO?

A: Updating is simple, just call the newer FEDIT. It will update the input files. Then continue with the corresponding FPLO.
For downgrading see Chapter 2.

Q: I started FEDIT and got the message

```
error(ReadPCTable): Cannot find entry definition file
'+fedit'!
It should be in the local tmp directory '+tmp'!
One can overwrite the location using option -ef filename.
Possible reason:
    1) the executable '...fplo...' is not running correctly
       or just does not exist!
    ....
```

A: Make sure that the FPLO executable (of the same version as FEDIT) is within the PATH (shell environment variable). Another possible source of the error is that the directory or some of the relevant files are write protected. If the FPLO executable is really not running (some linked libraries not found), just run FPLO by hand on command line and see what happens. Be aware of the fact that FEDIT will automatically try to execute the FPLO executable using the fully qualified name (with the 'version-release' and architecture-suffix, like `fplo22.00-62-x86_64`). It will not use the generic name `fplo`.

Q: How can I save memory?

A:

1. The number of **occupied bands** can be specified in the FEDIT main menu. Read the help screens for this variable. This does **not** work for CPA.
2. For large compounds the number of k -points in the Brillouin zone may be reduced. (Convergence with the number of k -points has always to be tested!)

Q: I want to make an antiferromagnetic calculation...

A: You will need two **spin sorts** and have to set the **initial polarization** to 't'. Then there is a submenu (press <SPACE> i), which allows to set the value of the **inital spinsplit**. Choose the same absolute value with opposite sign for the atoms, which are antiferromagnetically ordered. Set zero for atoms, which by symmetry must have zero moment. To assure zero total moment you may use the FSM method with a total **spin moment** of zero.

Note, that in the current implementation equivalent atoms with opposite magnetic moments have to be put on different Wyckoff positions. This is one of the rare occasions where the code does not make full use of symmetry.

Q: I want to plot the real space density or potential...

A: Use the grid-output sub-menu.

Obsolete since version 9.00. See description of file `.densgrid` in Chapter 3. The output is performed after the potential calculation in each iteration cycle (otherwise the potential is not yet defined). Use a plot program of your choice to visualize the data (e.g. `opendx`).

Q: FPLO cannot allocate memory.

A: Ask the system administrator to increase the limits of stack, heap and data size of a user job. FPLO jobs like other band structure programs need a lot of memory. The memory consumption is proportional to the number of k -points in the irreducible Brillouin zone, which in turn usually is proportional to the inverse of the number of atoms. You should check the number of k -points you really need to solve the physical question.

Use number of occupied bands to decrease memory usage!

Q: How can I see the progress of the calculation?

A: Assumed that you re-directed the FPLO output into a file, use the unix command:

```
grep "last deviation" fplo_outfile (for the self consistency)
grep "dev=" fplo_outfile (for the force iteration)
XFPLO -oi (set output file name in interface if needed)
```

Q: I want to use `less`, `grep`, `vi` or other unix tools to work with the FPLO-files having a name `'+...'`. It does not work!

A: Many (new) unix tools will interpret the `'+'` sign as an option flag. To use these tools with `'+'-files`, specify `./+file` instead `+file` on command line!

Q: I have started a spinpolarized calculation but the result is not spin polarized.

A: The symmetry between both spin directions has to be broken by hand. Set the initial polarization to `'t'`. You can set the amount of the initial split in the alternative submenu `Initial spinsplit`.

Attention: If there are already two spin sorts in the density file, the program will ask if the splitting shall be skipped (which is correct in most cases). However, in the case discussed in paragraph 16.4.1 a resplit was intended. If the job runs in background, it will abort as soon as the question has to be answered since a background job has normally no standard input to read from.

Q: I have started a spinpolarized calculation with initial spin split and the iteration jumps around.

A: May be the attractor regions for the polarized and for the non polarized solutions are selected in consecutive iteration steps and so it jumps. Depending on the given situation

1. one may try to increase the initial spin split.
2. the iteration-mixing may be too large. Got to alt-submenu `iteration` and decrease it.
3. use FSM to preconverge near the expected moment and release the FSM condition after convergence of the FSM calculation.
4. use FSM to scan to whole $E(M)$ curve (expensive, but sometimes the only way).

Bibliography

- [1] Technical report, <http://www.fplo.de>. 15.1
- [2] Axel D. Becke and Erin R. Johnson. A simple effective potential for exchange. *The Journal of Chemical Physics*, 124(22):221101, Jun 2006. 5.5.1
- [3] H. Eschrig. *Phys. Rev. B*, 80:104503, 2009. 6.1
- [4] David Koller, Fabien Tran, and Peter Blaha. Improving the modified becke-johnson exchange potential. *Physical Review B*, 85(15):155109, Apr 2012. 5.5.1, 5.1, 5.2, 5.3
- [5] Wei Ku. *Phys. Rev. Lett.*, 89:167204, 2002. 6.1
- [6] Wei Ku, Tom Berlijn, and Chi-Cheng Lee. Unfolding first-principles band structures. *Phys. Rev. Lett.*, 104(21):216401, May 2010. 15.1, 15.2.1
- [7] N. Marzari and D. Vanderbilt. *Phys. Rev. B*, 56:12847, 1997. 6.1
- [8] John P. Perdew and Yue Wang. Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B*, 45(23):13244–13249, Jun 1992. 5.5.1
- [9] R. Sakuma. *Phys. Rev. B*, 87:235109, 2013. 6.1
- [10] Fabien Tran and Peter Blaha. Accurate band gaps of semiconductors and insulators with a semilocal exchange-correlation potential. *Phys. Rev. Lett.*, 102(22):226401, Jun 2009. 5.5.1
- [11] Fabien Tran, Peter Blaha, and Karlheinz Schwarz. Band gap calculations with becke-johnson exchange potential. *Journal of Physics: Condensed Matter*, 19(19):196208, Apr 2007. 5.5.1
- [12] Erik van Heumen, Johannes Vuorinen, Klaus Koepnik, Freek Masee, Yingkai Huang, Ming Shi, Jesse Klei, Jeroen Goedkoop, Matti Lindroos, Jeroen van den Brink, and Mark S. Golden. Existence, character, and origin of surface-related bands in the high temperature iron pnictide superconductor *bafe2-xcoxas2*. *Phys. Rev. Lett.*, 106(2):027002, Jan 2011. 15.1, 15.6.3
- [13] Runzhi Wang. *Phys. Rev. B*, 90:165125, 2014. 6.1



Index

- band weights, 15, 19, 21, 25, 26, 77, 82, 143
basis, 31
- dHvA, 15, 25, 30, 94
- files
- +area_vs_angle_..., 25, 105
 - +atpot..., 27
 - +band..., 19, 21, 26, 95
 - +basis, 24
 - +bweights..., 19, 21, 26, 77, 78
 - +coeff, 25, 26
 - +dens..., 27
 - +dos..., 24, 25
 - +error, 26
 - +fcor..., 27
 - +fedit..., 16, 28
 - +fkval..., 27
 - +fval..., 27
 - +grid..., 28
 - +hamdata, 99, 102
 - +har..., 27
 - +idos..., 25
 - +imeps, 19, 25, 87–89, 140
 - +iso_..., 96
 - +iso_b*, 102
 - +iso_b..., 15, 19, 95, 97–100, 107
 - +iso_b..._p..._spin..., 25, 95
 - +isoergcache..., 19, 25, 94, 95, 98, 100, 101
 - +(i)ldos.site.nl, 25
 - +loi, 24
 - +mass_vs_angle_..., 25, 105
 - +plasmon, 28, 87, 88
 - +points, 27
 - +run, 26
 - +symmetry, 27
 - +tmp, 30
 - =.addwei, 21, 78, 97, 102
 - =.atcharge, 22, 143
 - =.basdef, 21, 36–38, 40, 41, 129, 140
 - =.basis, 26, 128, 129, 144
 - =.bwdef, 21, 26, 77, 78, 82, 97, 100–102
 - =.cmd, 22
 - =.coeff, 22, 143
 - =.dens, 19, 20, 26, 82, 83, 129, 144
 - =.densconvert, 18
 - =.densgrid, 23, 29, 144, 146
 - =.densmap, 21, 82, 83
 - =.dirac, 16
 - =.dmat_init, 20, 85, 86
 - =.in, 18, 20, 82, 107, 128, 142, 143
 - =.kp, 21, 26
 - =.ldos, 144
 - =.mol_charge, 143
 - =.pipe, 125
 - =.sym, 20, 142
 - =.unfold, 21
 - =.wandef, 99
 - =.xef, 22, 81, 96, 106
 - =.xftp, 22
 - =.xstr, 22, 81
 - area_vs_angle.cmd, 25, 30, 106
 - bravais.ps, 143
 - dHvAdata, 30, 106
 - dmatedit.ini, 30, 86
 - extrorbit_chain..._iphi...vtk, 106
 - grid_..., 29, 142
 - grid_...[net/general/cfg], 30
 - orbit_plane..._chain...vtk, 106
 - primitive.ps, 143
 - tmp, 30
 - wfdata..., 29, 142
- getting help, 1, 15, 83, 84
- initial spin polarization, 130, 145, 146
- mBJ, 45

optics, 15, 25, 28, 87, 91

programs

dirac, 15, 16, 28

dmatedit, 15, 20, 30, 44, 85, 86

faddwei, 15, 21, 77, 78, 99, 102

fdhva, 15, 25, 30, 94, 96, 102, 105–107

fdowngrad.pl, 18

fedit, 15

foptics, 15, 87, 89, 90

fout2in, 19, 21

fpiotest, 1, 15, 16

fplo, 15

grepfplo, 15, 44, 45, 73, 139

pyfplo, 36, 41, 110, 124, 140, 141

xfbp, 3, 15, 19, 22, 25–27, 30, 84, 87, 89, 90,
106, 139, 142

xfplo, 6, 15, 16, 20–22, 25, 26, 29, 56, 59, 61–
63, 81–83, 95, 96, 102, 106, 107, 141–143,
146

TI, 110

topological insulator, 110

unfolding, 21, 26, 111