TUTORIAL FOR MBXASPY
(Many-Body XAS with Python)

Yufeng Liang

Contents

Installation

(1) Shirley_XAS

Shirley_XAS is a software package for simulating x-ray absorption spectra using the excited-state-core-hole (XCH) approach and Shirley's optimal basis functions for band structure interpolation. It is modified based Quantum Espresso simulation package and maintained by David Prendergast's group at the Molecular Foundry. The Shirley_XAS provides the Kohn-Sham eigen-energies and wavefunctions as the input for the determinant formalism of x-ray spectra.

Check out the latest commit on the overlap branch (needs) if you have access to http://trac-foundry2.lbl.gov/git/Shirley:

```
git checkout overlap
git pull
make shirley
```

Or you may do module-load if you have access to local clusters of High-Performance Computing at Berkeley Lab:

```
module load shirley_overlap/QE4.3
```

(2) MBXASPY

Checkout the code on github:

```
git clone https://github.com/yufengliang/mbxaspy ~/mbxaspy
```

From now on you can follow my updates by using “git pull”. You will see many python source codes with postfix *.py are cloned to your specified directory.

Python 2.7+ is required.

Generate the pseudos and other auxiliary files

To perform a typical core-hole calculation, one needs pseudopotentials for the unexcited atom and also the core-excited one. Here is a list of codes that are relevant for generating pseudopotentials and other auxiliary files:

---

1. Vanderbilt Ultra-Soft Pseudopotential code (based on Fortran, with D. P. Prendergast’s developments for outputting intermediate data of atomic wavefunctions):

```
https://github.com/yufengliang/uspp-dgp
```

This is the main code for generating pseudopotentials.
After git-clone to local and enter the root folder:

```
cd uspp-dgp/Source
make clean
make
```

to compile the code.

2. shirley_QE4.3.overlap/upftools/uspp2upf.x (Fortran)

For converting the binary pseudo file into a *.UPF that can be read by Qespresso.

```
cd shirley_QE4.3.overlap/upftools
make clean
make
```

3. shirley_QE4.3.overlap/corerepair_dipole/corevalence_position.x (Fortran)

This code is for generating the single-body matrix elements for the position operator.

```
cd shirley_QE4.3.overlap/corerepair_dipole
make clean
make
```

4. mbxaspy/sij.py (PYTHON)

If you are only interested in elements that will NOT be core-excited, then the first two will do.

Here are steps for generating the core-hole pseudos (Using C 1s as an example):

1) Obtain sample input files from uspp-dgp/Work. In this folder, there are a number of examples for generating pseudos for some common elements. For example:

```
cd uspp-dgp/Work/006-C
ls
drwxr-x--- 2 yfliang yfliang   512 Dec 15 11:18 006-C-ca--bm
drwxr-x--- 2 yfliang yfliang   512 Dec 15 11:18 006-C-gpbe--bm
drwxr-x--- 2 yfliang yfliang  4096 Dec 15 11:28 006-C-gpbe--yufengl
drwxr-x--- 2 yfliang yfliang  4096 Dec 15 11:18 006-C-gpbe-1s1--yufengl
drwxr-x--- 2 yfliang yfliang   512 Dec 15 11:18 006-C-gpw-n-campos
```

The naming convention is: "006-C-ca—bm" means the carbon pseudo generated with functional "ca" (LDA) by the author "bm".

2) Make a new working directory for your ground-state atom from the sample.

You may:

```
cp -r 006-C-ca—bm 006-C-pbe-your_name
cd 006-C-pbe-your_name
make clean
```

If you cannot find the element you want in these samples, pick a similar element to start with.

3) Edit the input files as needed.

There are two input files: c_ae_s2p2.adat and c_ps.adat. The former is for all-electron calculation and the latter for pseudizing.

c_ae_s2p2.adat (all-electron)

```
  1     1    0    0    0    3       ifae,ifpsp,ifprt,ifplw,ilogd (5i5)
  2    1.50     -2.4       1.6   80 rlogd,emin,emax,nnt (3f10.5,i5)
  3    1.0d-11   1.0d-09   0.5   0  thresh,tol,damp,maxit (2e10.1,f10.5,i5)
  4    C                             title (a20)
  5    6.0    0.0       0.0         z,xion,exfact (f7.2,2f10.5)
  6   80.0       6.0      59.0      rmax,aasf,bbsf (3f10.5)
  7     3    0                      ncspvs,irel (2i5)
  8  100  2.       -20.0             nnlz,wwnl,ee (i4,f7.3,f14.6)
  9  200  2.        -1.0             nnlz,wwnl,ee (i4,f7.3,f14.6)
 10  210  2.        -0.5             nnlz,wwnl,ee (i4,f7.3,f14.6)
```

Instructions for the bolded parts:
line 4: must use *shorthand* notation for element names (case-sensitive !)
line 5: exfact defines the types of the functional: 0.0: LDA, 5.0: GGA-PBE (recommended)
line 6, 8-10: edit the z number and atomic shells as needed.

For more information about the all-electron (AE) part, please see:
http://physics.rutgers.edu/~dhv/uspp/uspp-736/Doc/INPUT_AE

c_ps.adat (generating pseudos (GEN))

```
 1     0    2    1    1    3         ifae,ifpsp,ifprt,ifplw,ilogd (5i5)
 2    1.50     -2.4       1.6   80   rlogd,emin,emax,nnt (3f10.5,i5)
 3    1.0d-11   1.0d-09   0.5   0    thresh,tol,damp,maxit (2e10.1,f10.5,i5)
 4   C                                  title (a20)
 5     1    2    2                     ncores,nvales,nang (3i5)
 6   10.0      20.0      40.0  10.0   besrmax,besemin,besemax,besde (4f10.5)
 7     3    0   0.8                     keyps,ifpcor,rinner (2i5,f10.5)
 8     4   1.0                         nbeta,rcloc (i5,f10.5)
 9    1.1       1.1       0.0           rc (3f10.5)
10     0    0  -1.2        2       lll,keyee,eeread,iptype (2i5,f10.5,i5)
11     0    0   0.2        2       lll,keyee,eeread,iptype (2i5,f10.5,i5)
12     1    0  -0.6        2       lll,keyee,eeread,iptype (2i5,f10.5,i5)
13     1    0   0.2        2       lll,keyee,eeread,iptype (2i5,f10.5,i5)
14     8  10.0                     npf,ptryc (i5,f10.5)
15     2    0   0.0        1       lloc,keyee,eloc,iploctype (2i5,f10.5,i5)
16     3    8  10.0                      ifqopt,nqf,qtryc (2i5,f10.5)
```

line 4: shorthand for title
line 5: ncores - # shells in the core, nvales - # valence shells, nang - #
line 10-13: edit these lines if generating for a different element (instructions as below)
line 16: must be 3 if GGA-PBE (exfact = 5.0 in the AE input.)

More instructions:
http://physics.rutgers.edu/~dhv/uspp/uspp-736/Doc/INPUT_GEN

4) Generate the pseudos:

Run:

```
make clean
make
../../../Bin/runatom.x c_ae_s2p2.adat c_ae_s2p2.out c_ae_s2p2.ae c_ae_s2p2.atwf c_ae_s2p2.logd
dummy
beginning execution pseudopotential program version  7.3.6
 beginning the all electron calculation
completed self-consistent cycle   1 delta =  0.2499440D+01
completed self-consistent cycle   2 delta =  0.1027510D+01
……
completed self-consistent cycle  37 delta =  0.1375697D-08
completed self-consistent cycle  38 delta =  0.7350098D-09
 all electron calculation completed
../../../Bin/runatom.x c_ps.adat c_ps.out c_ae_s2p2.ae c_ps.atwf c_ps.logd c_ps.uspp
beginning execution pseudopotential program version  7.3.6
 beginning the all electron calculation
completed self-consistent cycle   1 delta =  0.6194983D-10
 all electron calculation completed
 constructing ibeta =           1
```

```
 constructing ibeta =           2
 constructing ibeta =           3
 constructing ibeta =           4
 construction of qqq and ddd complete
 pseudizing qfuncs for beta= 1
 pseudizing qfuncs for beta= 2
 pseudizing qfuncs for beta= 3
 pseudizing qfuncs for beta= 4
 qfunc pseudized
 transformation of q and d complete
 transformation of qfunc complete
 transformation of qfcoef complete
 fourier transforming the qfunctions
 solving the schroedinger equation
 descreening the potential
 writing the pseudopotential to file
 fourier analysing pseudowavefunctions
 computing the logarithmic derivatives
 beginning search for possible ghost states
 eigensolution in bessel basis with cutoff 20.00
 eigensolution in bessel basis with cutoff 30.00
 eigensolution in bessel basis with cutoff 40.00
```

And you should be able to see these files:

```
-rw-r----- 1 yfliang yfliang   2237 Dec 15 13:33 Makefile
-rw-r----- 1 yfliang yfliang   1333 Dec 15 13:33 README
-rw-r----- 1 yfliang yfliang    652 Dec 15 13:40 c_ae_s2p2.adat
-rw-rw---- 1 yfliang yfliang  20572 Dec 15 14:03 c_ae_s2p2.ae
-rw-rw---- 1 yfliang yfliang   2040 Dec 15 14:03 c_ae_s2p2.logd
-rw-rw---- 1 yfliang yfliang  90430 Dec 15 14:03 c_ae_s2p2.out
-rw-r----- 1 yfliang yfliang   1104 Dec 15 13:47 c_ps.adat
-rw-rw---- 1 yfliang yfliang 141328 Dec 15 14:03 c_ps.atwf
-rw-rw---- 1 yfliang yfliang   4080 Dec 15 14:03 c_ps.logd
-rw-rw---- 1 yfliang yfliang 600089 Dec 15 14:03 c_ps.out
-rw-rw---- 1 yfliang yfliang  98088 Dec 15 14:03 c_ps.uspp
```

c_ae_s2p2.out and c_ps.out are the corresponding output for the c_ae_s2p2.adat and c_ps.out. You may check errors and warnings in these files if there is any. Intermediate data can also be found therein.

c_ps.uspp is the binary pseudo file and is not human-readable. Next we convert it into an UPF.

5) Convert the uspp to UPF

Run uspp2upf.x as follow:

```
shirley_QE4.3.overlapo/upftools/usppupf.x
```

and you'll see:

```
yfliang@edison09:~/PseudoP/uspp-dgp/Work/006-C/006-C-pbe-yufengl> ~/
shirley_QE4.3.overlap/upftools/uspp2upf.x
Input file > c_ps.uspp
Pseudopotential successfully read
Output PP file in UPF format :  c_ps.uspp.UPF
*** PLEASE TEST BEFORE USING!!! ***
review the content of the PP_INFO fields
```

6) Edit the pseudo.table to change the default pseudo or add a new one

Edit SYMBOL and MASS array if adding an element.

In the pbe block (**if** [[**-z** $1 **||** $1 **=** 'pbe']])

Edit the PSEUDO to add a new pseudo.

If you are not adding a pseudo for a core-excited atom, that's it. Below is for adding a new species of excited atom (update soon …)

****

Generate a ground-state and a core-excited pseudo for C routinely using executables in uspp-dgp. Let's say they are C.pbe-van.UPF and C.pbe-van-1s1.UPF respectively.

(2)
Generate the initial-state single-body matrix elements and name it as: C.pbe-van.pos

(3)
In the pseudo output file *_ps.out, we have access to transformation waves in the "**transformed atomic waves - davegp**" block. Copy every line **with digits only** in the block into a file called "valence-gs.dat" for the ground-state, and "valence-x.dat" for the excited state.

(4)
Run:

python /your/path/to/mbxaspy/sij.py valence-gs.dat valence-x.dat

(5)
Then you'll see a Sij.dat file in plain text, which is the overlap matrix for the transformation waves. Rename it as "C.pbe-van-1s1.sij" and place it in the XCH pseudo library so that mbxaspy can find it when the pseudo "C.pbe-van-1s1.UPF" is provided.

(6)
For sanity checks of the transformation waves, they are plot out to "**wave_gs.png**" and "**wave_x.png**". Use "eog file_name" to visualize them if on a cluster.

(7) If you run sij.py for the same atom, then you will see Sij.dat contains essentially the same values as Q_int in the atom's UPF.

Running MBXASPY

Part 1 Extend the previous shirley_xas calculations

(1) Run xas.sh, ref.sh, and ana.sh routinely. You can use the shirley_xas on the master branch without the many-body XAS development. The new many-body code is compatible with older calculations.

(2) On the overlap branch, a new script XAS_mbxas.sh is added. Cook up a script like this to run it:

```
. $SLURM_SUBMIT_DIR/Input_Block.in
 $SHIRLEY_ROOT/scripts/arvid/XAS_mbxas.sh
```

If the previous xas step has finished, this calculation is not expensive anyway and takes similar time as shirley_xas.x. So you may submit it to the debug queue in most cases when all Shirley interpolations are done.

This step generates:
*.eigval

eigenvalue file
*.eigvec

eigenvector file < B_i | nk > (transposed)
*.proj

projectors <beta | nk >
*.xmat

single-particle matrix elements <nk | r |phi_c>
overlap.dat

the overlap matrix < B_i | ~B_j >

This calculation will be done automatically for all specified excited atoms in the system and the ground state. These files will appear in the working directories for the excited atoms. overlap.dat is also produced for GS-to-GS transformation and ideally it should be an identity matrix.

You can even run mbxas.sh without any one of xas.sh, ref.sh, and ana.sh but you won't have access to *.xmat for all excited atoms (final states) which are not needed for many-body XAS, and you don't have access to energy shifts and fermi levels.

To automatically schedule all these calculations, you can alternatively checkout the xas_script on github:

git clone https://github.com/yufengliang/xas_script ~/xas_script

and run:

~/xas_script/setup_mbxas.sh

(on NERSC Edison)
~/xas_script/setup_mbxas_cori.sh

(on NERSC Cori)

to produce all 5 scripts: xas.sh, ref.sh, ana.sh, state.sh, and mbxas.sh.

Part 2 mbxaspy

(1) Cook up an input file (normally called mbxapy.in) like this:

```
# initial(ground)-state
path_i        =      'XAS/tio2/GS'  # path
mol_name_i    =      'tio2'                # file prefix
#nbnd_i       =                            # number of initial orbitals (not used now)

# final-state
path_f        =      'XAS/tio2/O5'    # path
#path_f       =      'XAS/tio2/GS'    # path
mol_name_f    =      'tio2.O05-FCH'              # file prefix
#mol_name_f    =      'tio2'               # file prefix
#nbnd_f       =                            # number of final orbitals (not used now)

xas_arg       =      5                    # number of k points along one direction
gamma_only    = True                                                                  #
Using gamma point only
nproc_per_pool =     2                    # number of procs used to process one (spin, k)
final_1p      = True                                  # Need one-body final-state spectrum
```

```
xi_analysis          = True                          # print out an analysis of the xi matrix
#do_paw_correction      = False
                                                                                                #
do PAW corrections or not
spec0_only = False   # only want one-body spectra

# spectral
maxfn = 1          # calculate up to f^(maxfn) order
I_thr = 1e-3       # throw away transitions that are below I_thr (fractional) of the strongest one.

ELOW               = -5
EHIGH              = 35
SIGMA              = 0.6
NENER               = 1000
ESHIFT_FINAL    = 517.976                  # *** Please note that this is the final ESHIFT
```

(2) Running mbxaspy

Command line:
Non-anaconda python:
python /your/path/to/mbxaspy/main.py < mbxaspy.in > mbxaspy.out &

Anaconda distribution of python (as on NERSC):
python /your/path/to/mbxaspy/main.py mbxaspy.in > mbxaspy.out &

MPI environment (Anaconda)
srun –n #proc python /your/path/to/mbxaspy/main.py mbxaspy.in > mbxaspy.out

On NERSC, be sure to unload python 2.7 and load 3.5-anaconda, which is more stable:

module load python/3.5-anaconda

For the f^(1) term for one (spin, k) tuple, the mbxaspy calculation is actually quite fast and you can run it on command line. It will take couples of minutes at most if the system contains < 100 atoms.

Multiple (spin, k) are supported and highly parallelizable but only individual spectra are printed out at the end because we still don't know how to combine them for multiple k-points.

Only one final-state is handled for each mbxaspy calculation. We can do more using scripts in future.

(3) Check mbxaspy.out for progress and error messages

(4) If the code runs successfully, in the end you will have for each (spin,k) tuple:

spec0_i.dat
one-body initial-state spectrum

spec0_f.dat
one-body final-state spectrum, if xas.sh is done and final-state xmat has been output.

spec_xas.dat
many-body XAS spectrum up to f^(maxfn) order

spec_xps.dat
many-body XPS spectrum up to f^(maxfn – 1) order

Columns in the spec files are: energy axis, total, x, y, z

(5) To check if the PAW corrections are done correctly, check test_xi_eig.png or test_xi_eig.dat, which contains all the eigenvalues of \xi and they should not be larger than 1.0.