Implementation of unfolding band structure in OpenMX

Chi-Cheng Lee
Institute for Solid State Physics, The University of Tokyo, Japan

OpenMX developer’s meeting           Kobe, Japan, Dec. 21st and 22nd, 2015.
Band structures between primitive and supercell unit cells

How about the spectral weight measured by experiments?

spectral weight at constant energy

Unfolding Band Structure
Spectral function, imaginary part of Green function, can be represented by a reference $|kj\rangle$ or $|kn\rangle$ basis for your purpose, where $|kj\rangle$ denotes the eigenstate of a conceptual system and $|kn\rangle$ the Wannier function. By inserting the known supercell eigenstates, which are obtained by DFT calculations, we have

$$A_{kn,kn}(\omega) = \sum_{KJ} <kn|KJ><KJ|A(\omega)|KJ><KJ|kn>$$

$$A_{kn,kn}(\omega) = \sum_{RN} |\langle kn|RN\rangle|^{2} A_{KJ,KJ}(\omega)$$

$$\langle kn|KJ\rangle = \sum_{RN} \langle kn|RN\rangle\langle RN|KN\rangle\langle KN|KJ\rangle$$

$$= \sqrt{1/Ll} \sum_{RN} e^{i(K-k)\cdot R} e^{-ik\cdot r(N)} \delta_{n,n'(N)} \langle KN|KJ\rangle$$

$$= \sqrt{L/l} \sum_{N} e^{-ik\cdot r(N)} \delta_{n,n'(N)} \delta_{[k],K} \langle KN|KJ\rangle$$

The information are hidden in the structure factor and wavefunctions. The only effort you need to do is to relabel RN by rn, others by DFT.

Unfolding first-principles band structures
Unfolding method for first-principles LCAO electronic structure calculations
Chi-Cheng Lee, Yukiko Yamada-Takamura, and Taisuke Ozaki

Idea of unfolding band structure: real-space assignment with LCAO basis

Overlap matrix $S$ needs to be considered!
$S$ knows the broken translational symmetry!

$$A_{kj,kj} = \sum_{mnK} S_{nm}^{-1}(k) \langle km|KJ\rangle A_{KJ,KJ} \langle KJ|kn\rangle$$

$$A_{kj,kj}(\omega) = \frac{L}{l} \sum_{KG} \delta_{k-G,K} W_{KJ} A_{KJ,KJ}(\omega)$$

What we need to do is
$$R \to R + r_0(M) \text{ and } M \to m'(M)$$

to get the phase, everything else can be obtained by OpenMX.

unfolding is performed eigenstate by eigenstate
unfolding is not performed

perfect supercell

conventional unit cell (two Fe atoms)

black curves are from primitive cell calculation
conventional unit cell (two Fe atoms)

unfolding is performed

perfect supercell

(Spin up channel)

black curves are from primitive cell calculation
unfolding is performed

bcc Fe

conventional unit cell (two Fe atoms)

1  Fe  0.000  0.000  0.000
2  Fe  0.510  0.510  0.490

lattice constant: 2.87 Angstrom

(Spin up channel)

phonon effect is tiny on band structure
bcc Fe

unfolding is performed

1 Fe 0.000 0.000 0.000
2 Fe 0.551 0.546 0.465

lattice constant: 2.87 Angstrom

conventional unit cell (two Fe atoms)

(Spin up channel)

stronger translation symmetry breaking
The reference cell is conceptual
Silicene on ZrB$_2$ with incommensurate primitive cells (six Si atoms per unit cell)

Planar-like phase

$\sqrt{3} \times \sqrt{3}$-reconstructed silicene

commensurate cell
Silicene

Free-standing planar-like silicene

six Si atoms per unit cell is needed in the supercell

E = 1eV below the Fermi energy

(i) Planar-like phase
(j)

(k)
(l)

The strategy is to revise the Band_DFT_MO.c since we already have the LCAO coefficients at the provided $k$ points.
Input file

Unfolding.fileout on Turn on unfolding.

\begin{verbatim}
<Unfolding.ReferenceVectors
  1.4350  1.4350 -1.4350
  -1.4350  1.4350  1.4350
  1.4350 -1.4350  1.4350
Unfolding.ReferenceVectors>

<Unfolding.Referenceorigin
  -0.1 -0.2 -0.3
Unfolding.Referenceorigin>

<Unfolding.Map
  1 1
  2 1
Unfolding.Map>

Unfolding.LowerBound -9.0
Unfolding.UpperBound  9.0

Unfolding.desired_totalnkpt 40

Unfolding.Nkpoint 5

<Unfolding.kpoint
  0 0 0
  -0.5 0.5 0.5
  0 0 0.5
  0 0 0
  0.25 0.25 0.25
Unfolding.kpoint>
\end{verbatim}

Define reference lattice vectors. The form and unit are the same as those in Atoms.UnitVectors.

Define the origin of the reference unit cell. Together with the lattice vectors defined above, the reference cell is uniquely defined.

Define mapping between supercell atoms and reference-cell atoms by integers. Here both supercell atom 1 and atom 2 are assigned to the reference-cell atom 1. This is possible because they belong to different reference lattice vectors.

Define the energy window to perform the unfolding (in eV, to \(E_F\)).

Provide the desired total number of \(k\) points along the paths.

Provide the number of high symmetry \(k\) points.

List the high symmetry \(k\) points.

\begin{verbatim}
<openmx_common.h>
  double **unfold_abc;
  double *unfold_origin;
  int *unfold_mapN2n;
  double unfold_lbound,unfold_ubound;
  int unfold_fileout;
  int unfold_Nkpoint;
  int unfold_nkpts;
  double **unfold_kpoint;
\end{verbatim}
Reading input information and setting up the default values are done here.

The default values are set to unfold the bands from the calculating Brillouin zone to the same zone. The way for unfolding everything to itself simply provide the orbital weight on the band structure.

```c
# Input_std.c

double **unfold_abc;

defined to the original abc

double *unfold_origin;

defined to (-0.00001, -0.00001, -0.00001)

int *unfold_mapN2n;

defined to
for (i=0; i<atomnum; i++) unfold_mapN2n[i]=i;

double unfold_lbound, unfold_ubound;

defined to [-10 eV, 10 eV]

int unfold_Nkpoint;

number of high symmetry k points is defined to 0

int unfold_nkpts;

number of desired k points is defined to 0
```
if (unfold_fileout==1 && unfold_Nkpoint>0 && (Solver==2 || Solver==3)) {
    if (Cnt_switch==0){
        Unfolding_Bands(unfold_Nkpoint, unfold_kpoint, SpinP_switch, H, iHNL, OLP[0]);
    }
    else {
        Unfolding_Bands(unfold_Nkpoint, unfold_kpoint, SpinP_switch, CntH, iCntHNL, CntOLP[0]);
    }
}

Migration from <Band_DFT_MO.c> to <Band_Unfolding.c>

void Unfolding_Bands( int nkpoint, double **kpoint,  
                         int SpinP_switch,          
                         double ******nh,          
                         double ******ImNL,         
                         double ******CntOLP)  
{
    if (SpinP_switch==0 || SpinP_switch==1){
        Band_DFT_MO_Col
        Unfolding_Bands_Col(nkpoint, kpoint, SpinP_switch, nh, CntOLP);
    }  
    else if (SpinP_switch==3){
        Band_DFT_MO_NonCol
        Unfolding_Bands_NonCol(nkpoint, kpoint, SpinP_switch, nh, ImNL, CntOLP);
    }
}
void buildMapRlist();

double***** Elem;

int** Rlist;

size of Rlist is (#R,3)

S(0, Atom; R, Atom’) = S(R, Atom, Atom’)
= Elem(R,Atom’,Atom’,Orbital,Orbital’)

double **unfold_abc;
double *unfold_origin;
int *unfold_mapN2n;

assign each supercell RN with reference-cell r

int*** tabr4RN;
tabr4RN[iR][iAtom][0]= r0
tabr4RN[iR][iAtom][1]= r1
tabr4RN[iR][iAtom][2]= r2

int*** rnmap;  buildrnmap(mapN2n)

assign each reference-cell atom to supercell R Atom

int** rlist;

size of rlist is (#r,3)

size of rnmap is (#r,#atom,2)
rnmap(i,j,0)=iR
rnmap(i,j,1)=iAtom
Input file
Unfolding.desired_totalnkpt 40
Unfolding.Nkpoint 5
Unfolding.kpoint
0 0 0
0.5 0.5 0.5
0 0 0.5
0.25 0.25 0.25
Unfolding.kpoint

void determine_kpts(const int nk, double** klist);

int kloopi,kloopj;
double kpt0,kpt1,kpt2;
for (kloopi=0; kloopi<nkpoint; kloopi++)
    for (kloopj=0; kloopj<np[kloopi]; kloopj++) {
        ...
        store EIGEN[spin][j1] to kj_e and C[spin][j1][i1] to kj_v
        ...
    }
Coding (currently working on source3.744)

Supercell overlap matrix elements

Supercell LCAO coefficients

Definition of the reference cell

Chi-Cheng Lee, Yukiko Yamada-Takamura, and Taisuke Ozaki

3.5. Orbital contribution

With the obtained unfolding formula, it is also very interesting to see how each basis function contributes to the unfolded spectral weight. The decomposition to each contribution may be possible by defining $W_{KJM}^k$ as follows:

$$ W_{KJM}^k = \frac{L}{l} \sum_G \delta_{k-G,K} C_M^{KJ} \sum_{rN} e^{ik(r-r')(M)} \times C_N^{KJ*} S_{0N,rm(M)}. $$

(26)

Then, the spectral function can be written as

$$ A_{kj,kj}(\omega) = \sum_K A_{KJ,kj}(\omega) \sum_M W_{KJM}^k. $$

(27)
# Coding (currently working on source3.744)

```c
for (j=0; j<atomnum; j++) for (k=0; k<atomnum; k++) for (l=0; l<Norbperatom[j]; l++)
    tmpelem[j][k][l][m]=Cmul(Conjg(kj_v[countkj_e][j][l]), kj_v[countkj_e][k][m]);

int NA,ir,MA,MO,NO;
dcomplex dtmp;
for (NA=0; NA<atomnum; NA++) {
    int n=unfold_mapN2n[NA];
    int r0x=tabr4RN[0][NA][0];
    int r0y=tabr4RN[0][NA][1];
    int r0z=tabr4RN[0][NA][2];
    r0[0]=r0x*a[0]+r0y*b[0]+r0z*c[0];
    r0[1]=r0x*a[1]+r0y*b[1]+r0z*c[1];
    r0[2]=r0x*a[2]+r0y*b[2]+r0z*c[2];
    dcomplex phase1=Cexp(Complex(0.,-dot(K,r0)));
    for (ir=0; ir<nr; ir++) {
        if (rnmap[ir][n][1]==-1) continue;
        r0=rlist[ir][0]*a[0]+rlist[ir][1]*b[0]+rlist[ir][2]*c[0];
        r1=rlist[ir][0]*a[1]+rlist[ir][1]*b[1]+rlist[ir][2]*c[1];
        r2=rlist[ir][0]*a[2]+rlist[ir][1]*b[2]+rlist[ir][2]*c[2];
        dcomplex phase2=Cmul(phase1,Cexp(Complex(0.,dot(K,r))));
        for (MA=0; MA<atomnum; MA++) {
            if (Elem[rnmap[ir][n][0]][MA][rnmap[ir][n][1]][0][0]<-99999.) continue;
            for (MO=0; MO<Norbperatom[MA]; MO++) for (NO=0; NO<Norbperatom[NA]; NO++) {
                dtmp=RCmul(Elem[rnmap[ir][n][0]][MA][rnmap[ir][n][1]][MO][NO],tmpelem[MA][NA][MO][NO]);
                weight[NA][NO]=Cadd(weight[NA][NO],Cmul(phase2,dtmp));
            }
        }
    }
}
```

Efficiency can be improved later.

```
output contribution of each orbital for each |kj>: weight[Atom][Orbital]
or total weight:
```

```c
double sumallorb=0.;
for (j=0; j<atomnum; j++) for (k=0; k<Norbperatom[j]; k++) sumallorb+=weight[j][k].r;
```
Unfolding_Bands_NonCol(nkpoint, kpoint, SpinP_switch, nh, ImNL, CntOLP);

    kj_v[countkj_e][Gc_AN-1][iorb]=Complex(H[i1][j1].r,H[i1][j1].i);
    kj_v1[countkj_e++][Gc_AN-1][iorb]=Complex(H[i1+n][j1].r,H[i1+n][j1].i);

... for (j=0; j<atomnum; j++) for (k=0; k<atomnum; k++) for (l=0; l<Norbperatom[j]; l++) for (m=0; m<Norbperatom[k]; m++)
    tmpelem[j][k][l][m]=Cmul(Conjg(kj_v[countkj_e][j][l]),kj_v[countkj_e][k][m]);

... for (j=0; j<atomnum; j++) for (k=0; k<atomnum; k++) for (l=0; l<Norbperatom[j]; l++) for (m=0; m<Norbperatom[k]; m++)
    tmpelem1[j][k][l][m]=Cmul(Conjg(kj_v1[countkj_e][j][l]),kj_v1[countkj_e][k][m]);

... for (MA=0; MA<atomnum; MA++) {
    if (Elem[rnmap[ir][n][0]][MA][rnmap[ir][n][1]][0][0]<-99999.) continue;
    for (MO=0; MO<Norbperatom[MA]; MO++) for (NO=0; NO<Norbperatom[NA]; NO++)
        dtmp=RCmul(Elem[rnmap[ir][n][0]][MA][rnmap[ir][n][1]][MO][NO],tmpelem[MA][NA][MO][NO]);
    dtmp1=RCmul(Elem[rnmap[ir][n][0]][MA][rnmap[ir][n][1]][MO][NO],tmpelem1[MA][NA][MO][NO]);
    weight[NA][NO]=Cadd(weight[NA][NO],Cmul(phase2,dtmp));
    weight1[NA][NO]=Cadd(weight1[NA][NO],Cmul(phase2,dtmp1));
}

for (j=0; j<atomnum; j++) for (k=0; k<atomnum; k++) sumallorb+=weight[j][k].r;
for (j=0; j<atomnum; j++) for (k=0; k<atomnum; k++) sumallorb+=weight1[j][k].r;
fprintf(fp_EV1,"%f %f %10.7f\n",kdis,kj_e[countkj_e]*eV2Hartree,sumallorb/coe);
for (j=0; j<atomnum; j++) { for (k=0; k<atomnum; k++) fprintf(fp_EV1,"%e ",(weight[j][k].r+weight1[j][k].r)/coe);
fprintf(fp_EV1,"\n");
}

\*\*\*, we have two components in non-collinear case
Input file

Unfolding.desired_totalnkpt 40

Unfolding.Nkpoint 5

<Unfolding.kpoint
  0 0 0
-0.5 0.5 0.5
  0 0 0.5
  0 0 0
  0.25 0.25 0.25
Unfolding.kpoint>

Standard output:

**********************************************
Band_Unfolding is switched on
**********************************************

The number of selected k points is 38.
( 11 8 8 10 1 )

< ka kb kc >
0.000000 0.000000 0.000000
-0.045455 0.045455 0.045455
-0.090909 0.090909 0.090909
-0.136364 0.136364 0.136364
-0.181818 0.181818 0.181818
-0.227273 0.227273 0.227273
-0.272727 0.272727 0.272727
-0.318182 0.318182 0.318182
-0.363636 0.363636 0.363636
-0.409091 0.409091 0.409091
-0.454545 0.454545 0.454545
-0.500000 0.500000 0.500000
-0.437500 0.437500 0.500000
-0.375000 0.375000 0.500000
-0.312500 0.312500 0.500000
-0.250000 0.250000 0.500000
-0.187500 0.187500 0.500000
-0.125000 0.125000 0.500000
-0.062500 0.062500 0.500000
  0.000000 0.000000 0.500000
  0.000000 0.000000 0.437500
  0.000000 0.000000 0.375000
  0.000000 0.000000 0.312500
  0.000000 0.000000 0.250000
  0.000000 0.000000 0.187500
  0.000000 0.000000 0.125000
  0.000000 0.000000 0.062500
  0.000000 0.000000 0.000000
  0.025000 0.025000 0.025000
  0.050000 0.050000 0.050000
  0.075000 0.075000 0.075000
  0.100000 0.100000 0.100000
  0.125000 0.125000 0.125000
  0.150000 0.150000 0.150000
  0.175000 0.175000 0.175000
  0.200000 0.200000 0.200000
  0.225000 0.225000 0.225000
  0.250000 0.250000 0.250000
output file: case.out

***UNFOLDING CALCULATION***

Unfolded weights at specified k points are stored in case.unfold_totup(dn)
Individual orbital weights are stored in case.unfold_orbup(dn)
The format is: k_dis(Bohr-1) energy(eV) weight
The sequence for the orbital weights in case.unfold_orbup(dn) can be found below.

Unfolded weights at specified k points are stored in case.unfold_tot
Individual orbital weights are stored in case.unfold_orb
The format is: k_dis(Bohr-1) energy(eV) weight
The sequence for the orbital weights in case.unfold_orb can be found below.

```
1  Fe 0 s
   1 s
   2 s
   0 px
   0 py
   0 pz
   1 px
   1 py
   1 pz
   2 px
   2 py
   2 pz
   0 d3z^2-r^2
   0 dx^2-y^2
   0 dxy
   0 dxz
   0 dyz
   1 d3z^2-r^2
   1 dx^2-y^2
   1 dxy
   1 dxz
   1 dyz
...
2  Fe 0 s
   1 s
   2 s
   0 px
   0 py
   0 pz
   1 px
   1 py
   ...
```
case.unfold_totup

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000000</td>
<td>-8.207744</td>
<td>1.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>-4.593939</td>
<td>0.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>-4.593938</td>
<td>0.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>-2.231539</td>
<td>1.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>-2.231538</td>
<td>1.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>-2.231536</td>
<td>1.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>-1.012023</td>
<td>1.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>-1.012021</td>
<td>1.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>0.088024</td>
<td>0.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>0.088025</td>
<td>0.0000000</td>
</tr>
<tr>
<td>0.0000000</td>
<td>0.088027</td>
<td>0.0000000</td>
</tr>
<tr>
<td>0.105319</td>
<td>-8.041093</td>
<td>1.0000000</td>
</tr>
<tr>
<td>0.105319</td>
<td>-4.523466</td>
<td>0.0000000</td>
</tr>
</tbody>
</table>

Grace

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000000</td>
<td>-8.207744</td>
<td>5.196924e-01</td>
</tr>
<tr>
<td>0.105319</td>
<td>-8.041093</td>
<td>-2.10113e-17</td>
</tr>
</tbody>
</table>

Gnuplot

```plaintext
plot 'case.unfold_totup' using 1:2:($3)*0.05 with circles
```

be cautious with negative values in plotting the weight
Then you can, for example,

1. Prepare the figures together with the manuscript.
3. Submit to a journal.
4. Keep your fingers crossed.

Thank you for your attention!