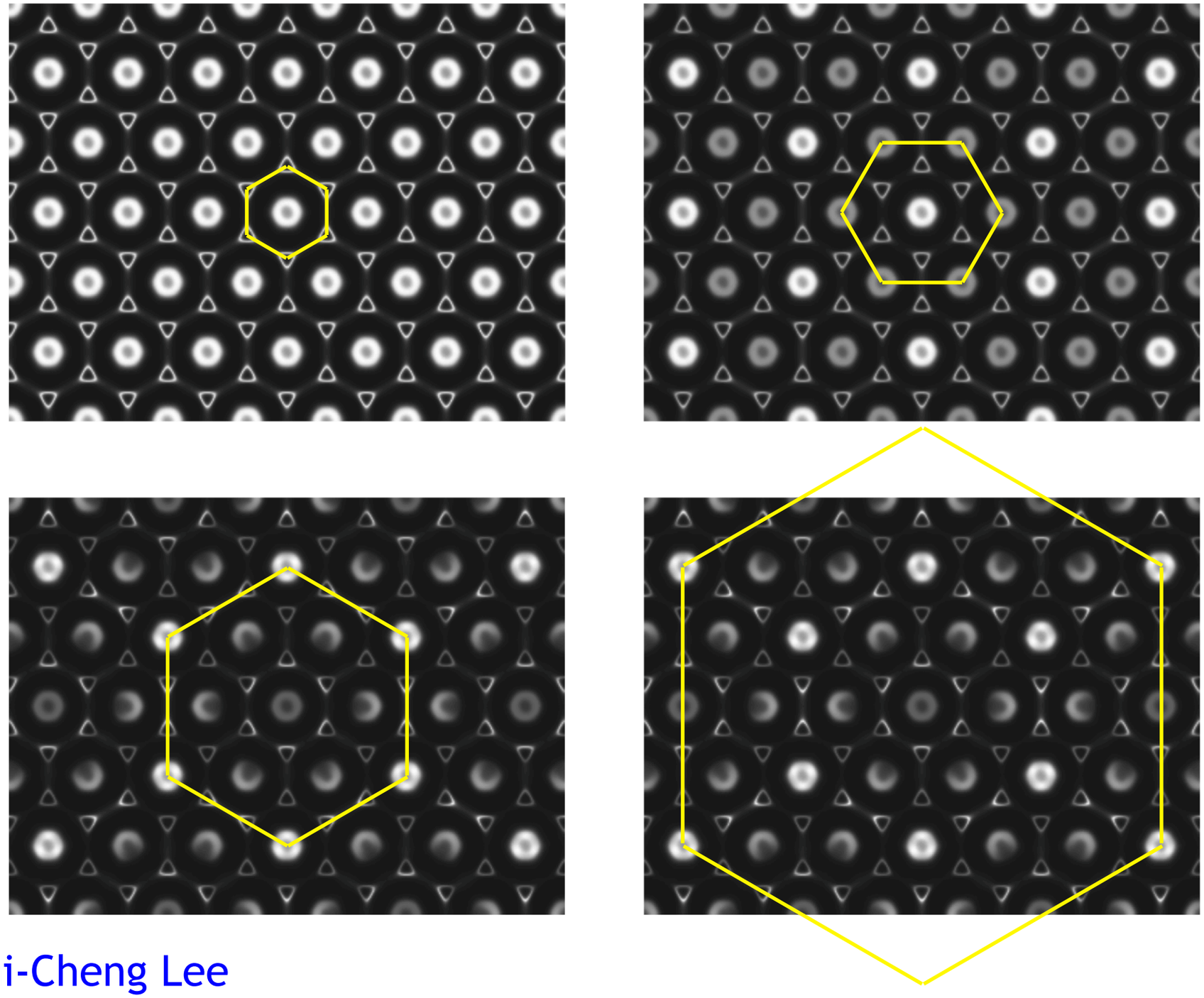
Implementation of unfolding band structure in OpenMX

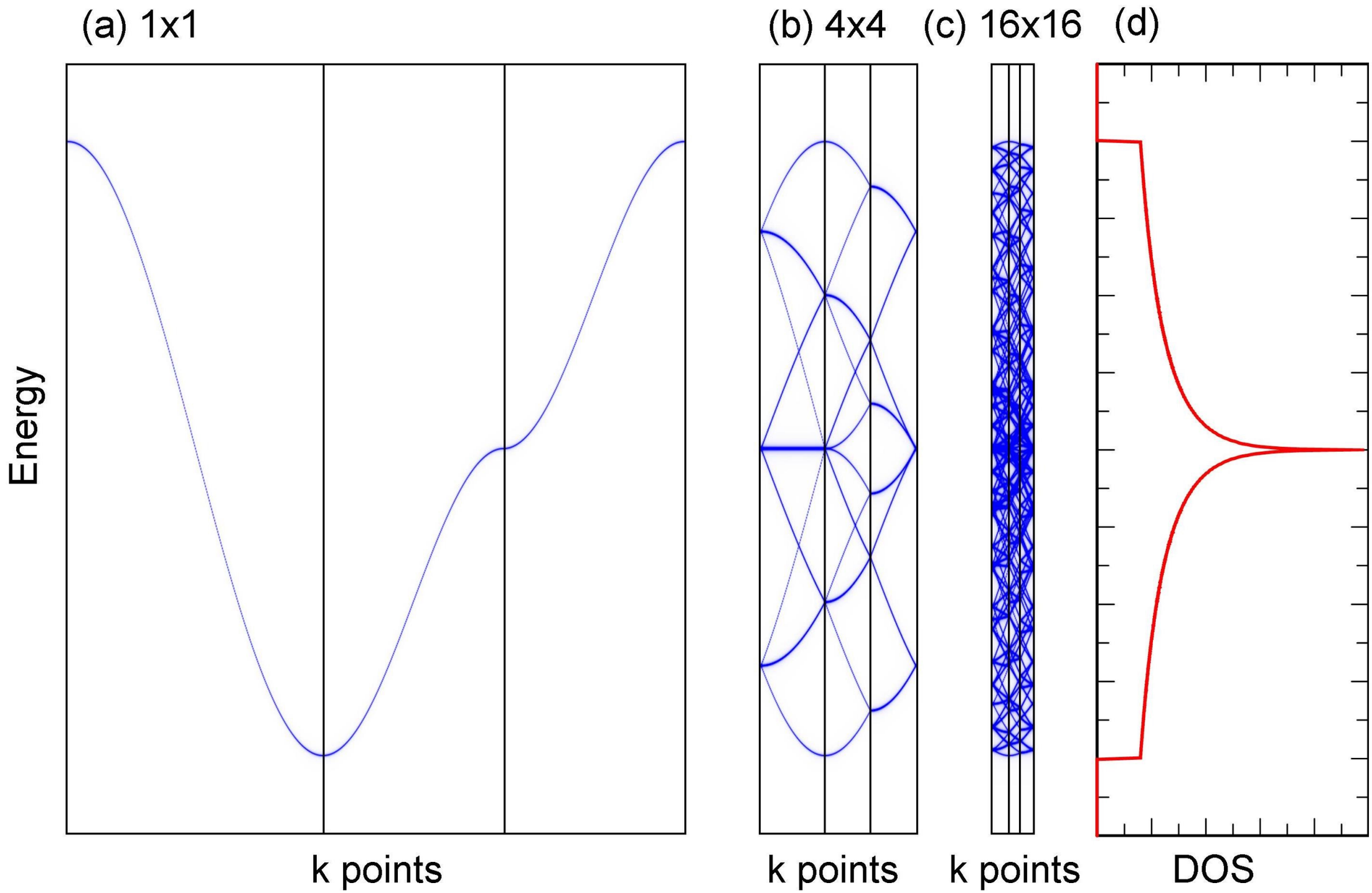


Chi-Cheng Lee
Institute for Solid State Physics, The University of Tokyo, Japan

OpenMX developer's meeting **Kobe, Japan, Dec. 21st and 22nd, 2015.**

Band Folding

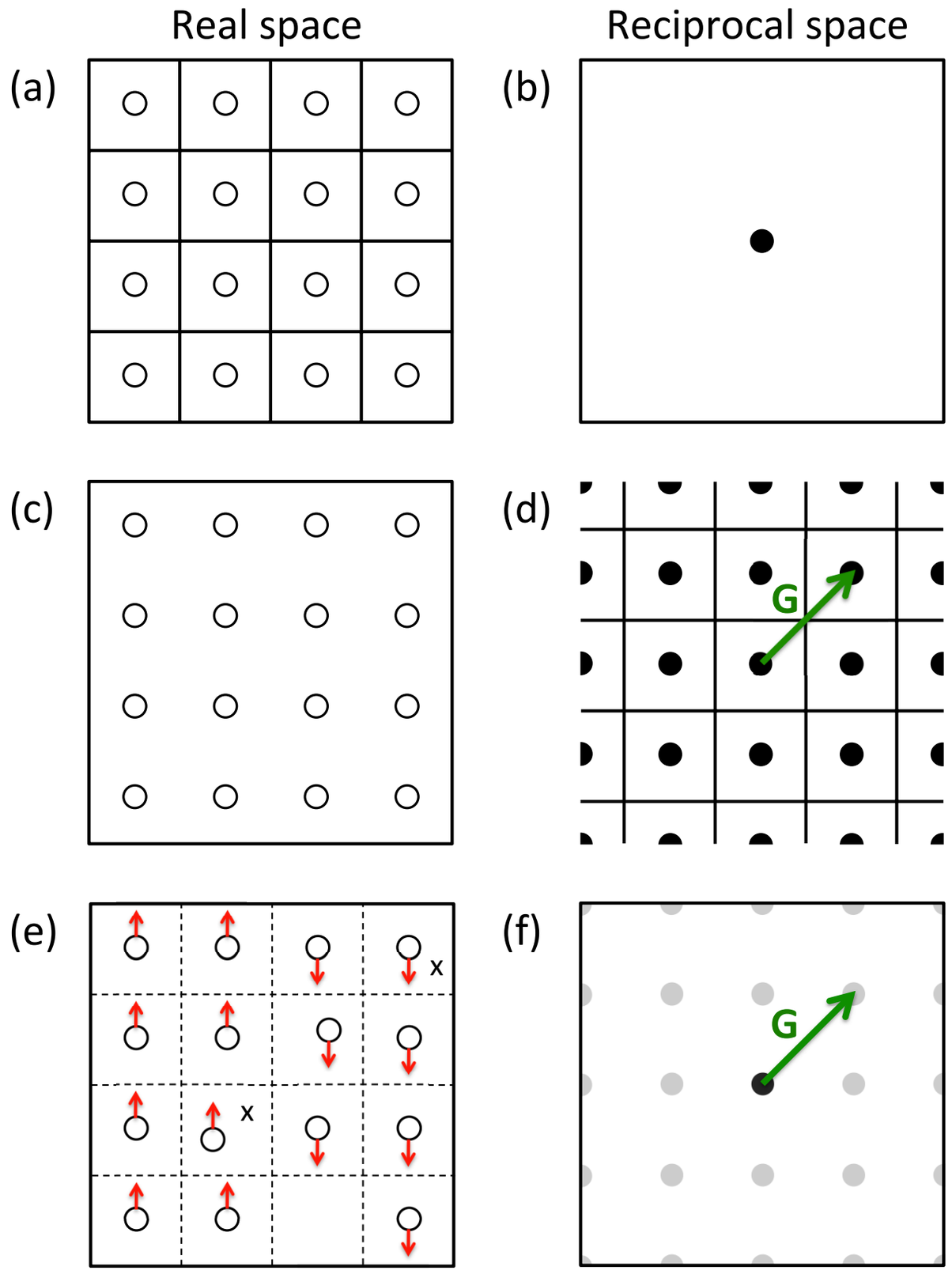
Band structures between primitive and supercell unit cells



Phys. Rev. Lett. **104**, 216401 (2010).

How about the spectral weight measured by experiments?

spectral weight at constant energy



J. Phys.: Condens. Matter **25**, 345501 (2013).

Unfolding Band Structure

Idea of unfolding band structure: real-space assignment with Wannier function

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

$$A_{kn,kn}(\omega) = \Sigma_{KJ} \langle kn|KJ\rangle\langle KJ|A(\omega)|KJ\rangle\langle KJ|kn\rangle$$

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

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

$$= \sum_{RN} \langle kn|R + r(N), n'(N)\rangle\langle RN|KN\rangle\langle KN|KJ\rangle$$

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

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

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
Wei Ku, Tom Berlijn, and Chi-Cheng Lee, Phys. Rev. Lett. **104**, 216401 (2010).

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

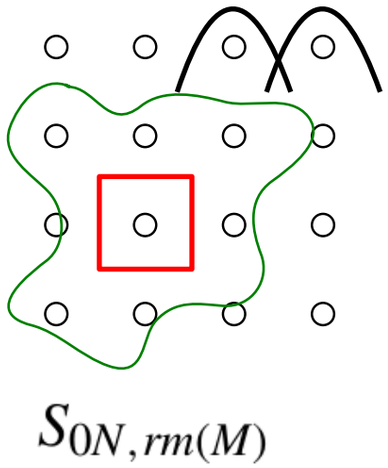
$$A_{kj,kj} = \sum_{mnK} S^{-1}_{nm}(k) \langle km|KJ\rangle A_{KJ,KJ} \langle KJ|kn\rangle$$

100% at ε_{kj}

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

$$W_{KJ} = \sum_{MNr} \mathrm{e}^{\mathrm{i}k(r-r'(M))} C^{KJ}_{M} C^{KJ*}_{N} S_{0N,rm(M)}$$

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



real space

$S_{0N,rm(M)}$

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.

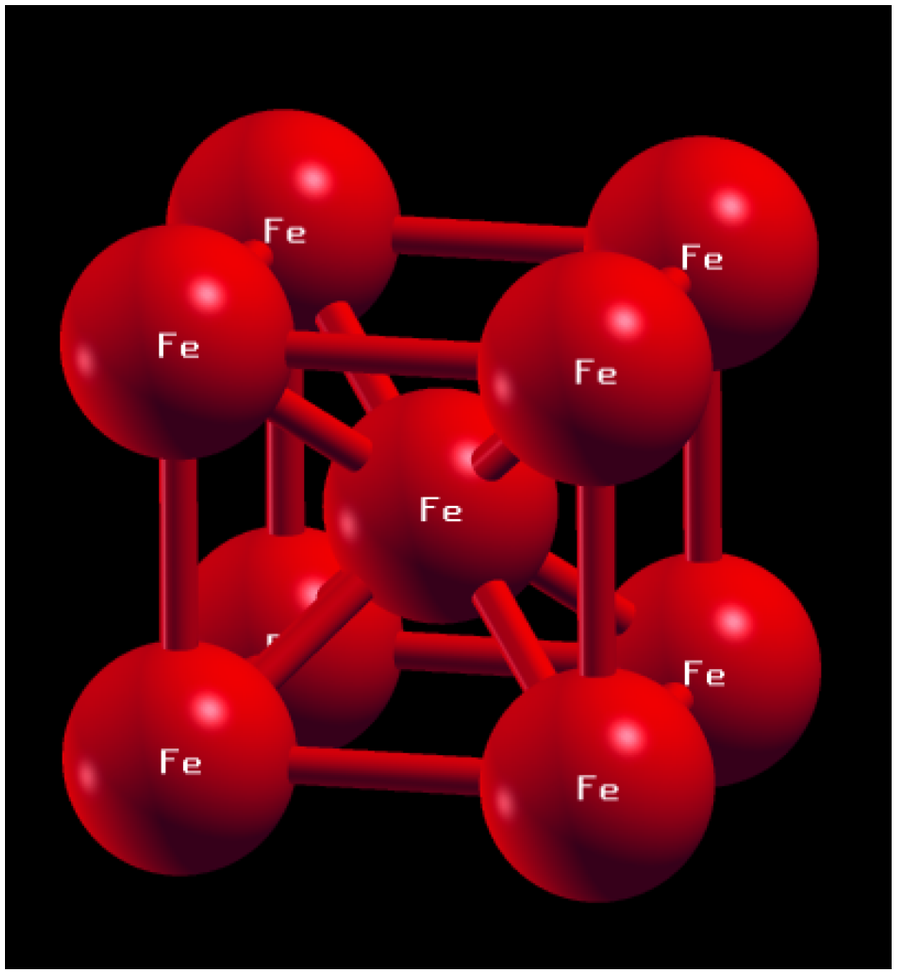
unfolding is performed eigenstate by eigenstate

Unfolding method for first-principles LCAO electronic structure calculations
Chi-Cheng Lee, Yukiko Yamada-Takamura, and Taisuke Ozaki
J. Phys.: Condens. Matter **25**, 345501 (2013).

Examples

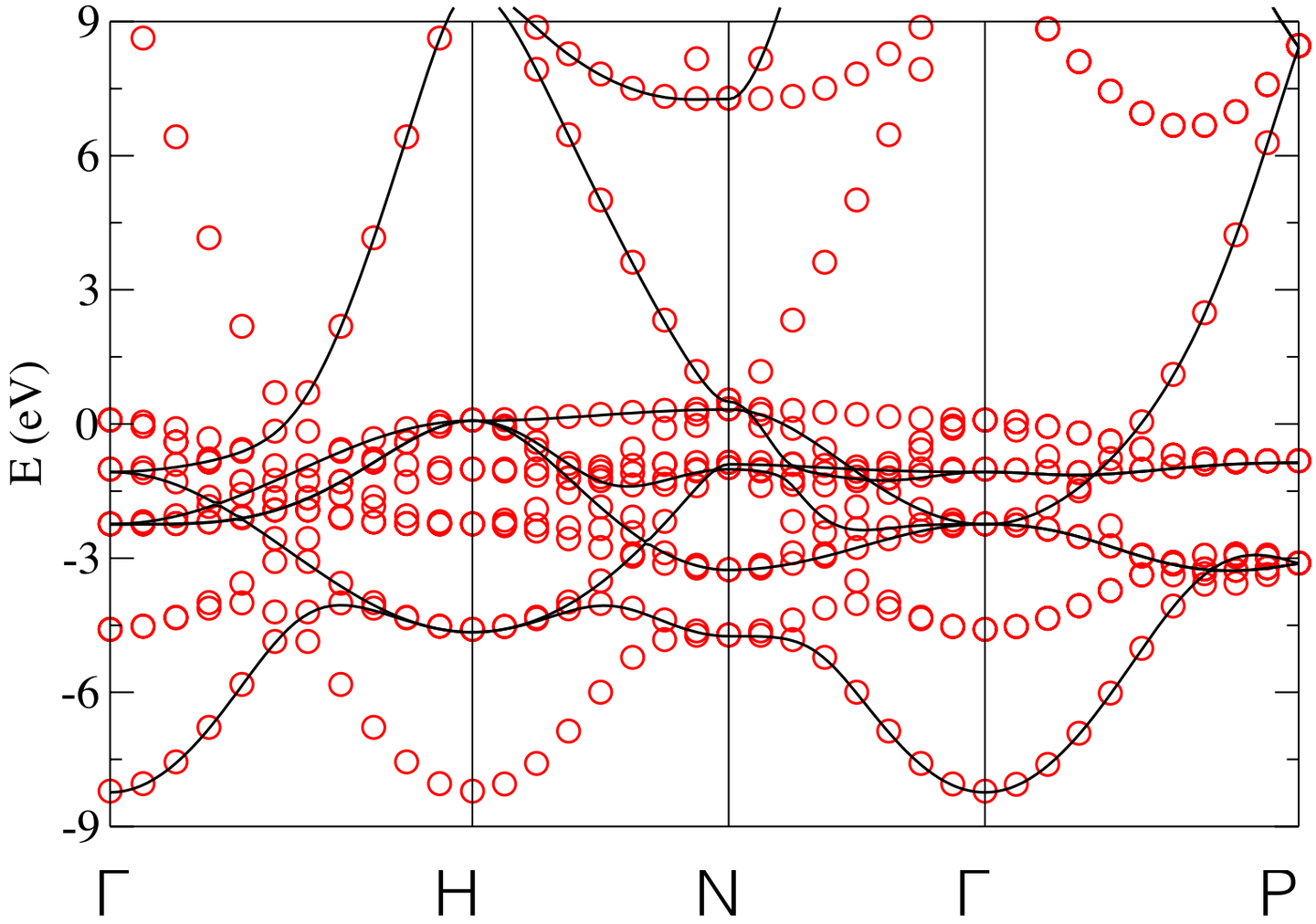
bcc Fe

unfolding is not performed



perfect supercell

conventional unit cell (two Fe atoms)

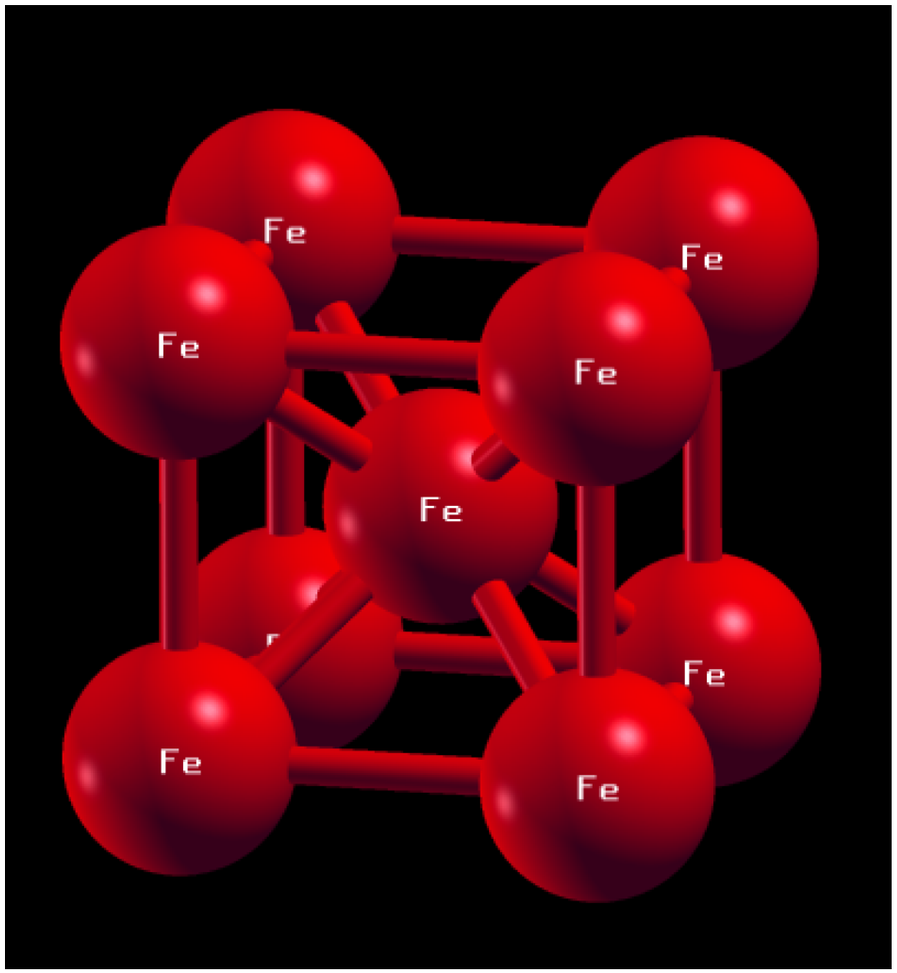


(Spin up channel)

black curves are from primitive cell calculation

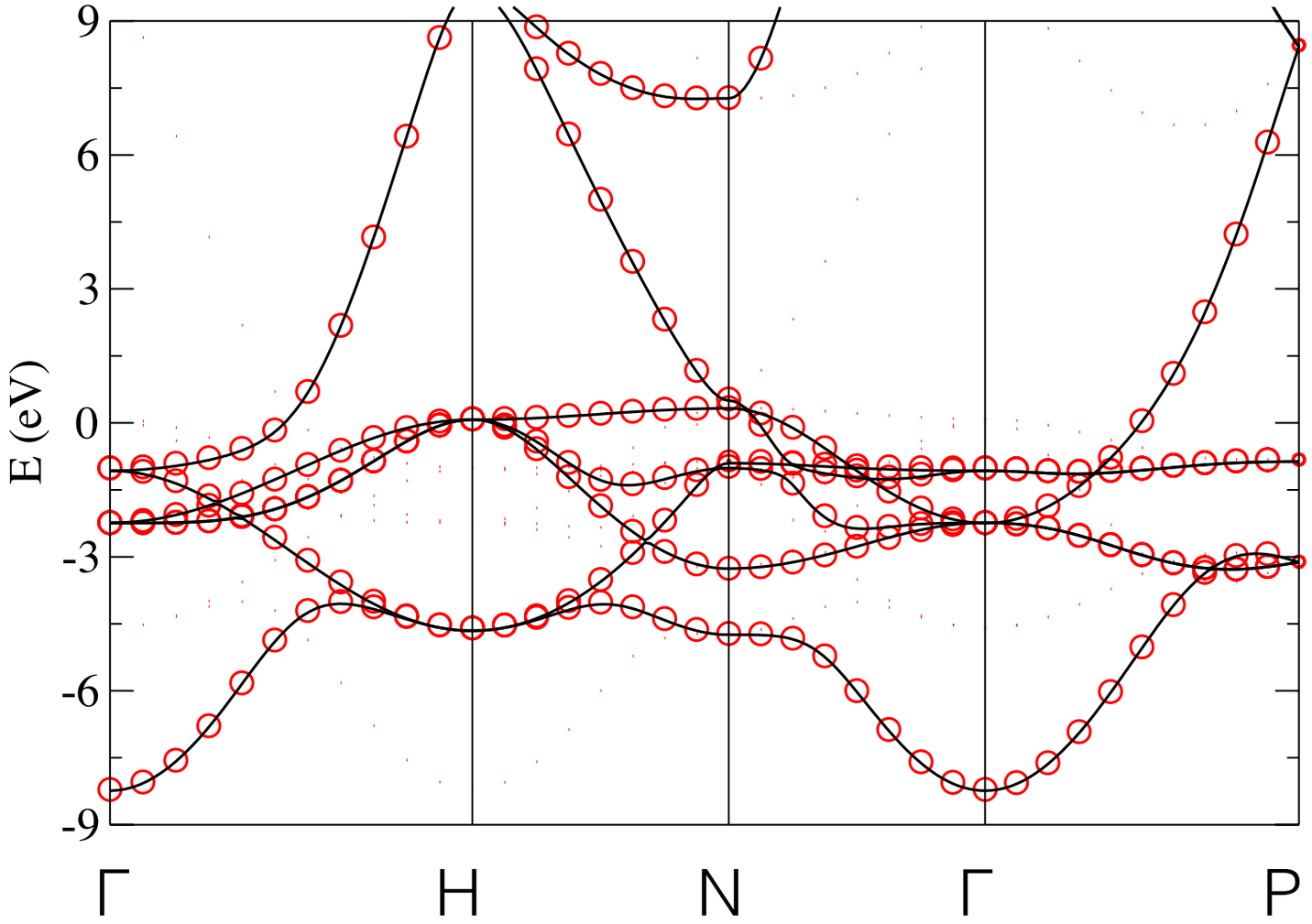
bcc Fe

unfolding is performed



perfect supercell

conventional unit cell (two Fe atoms)

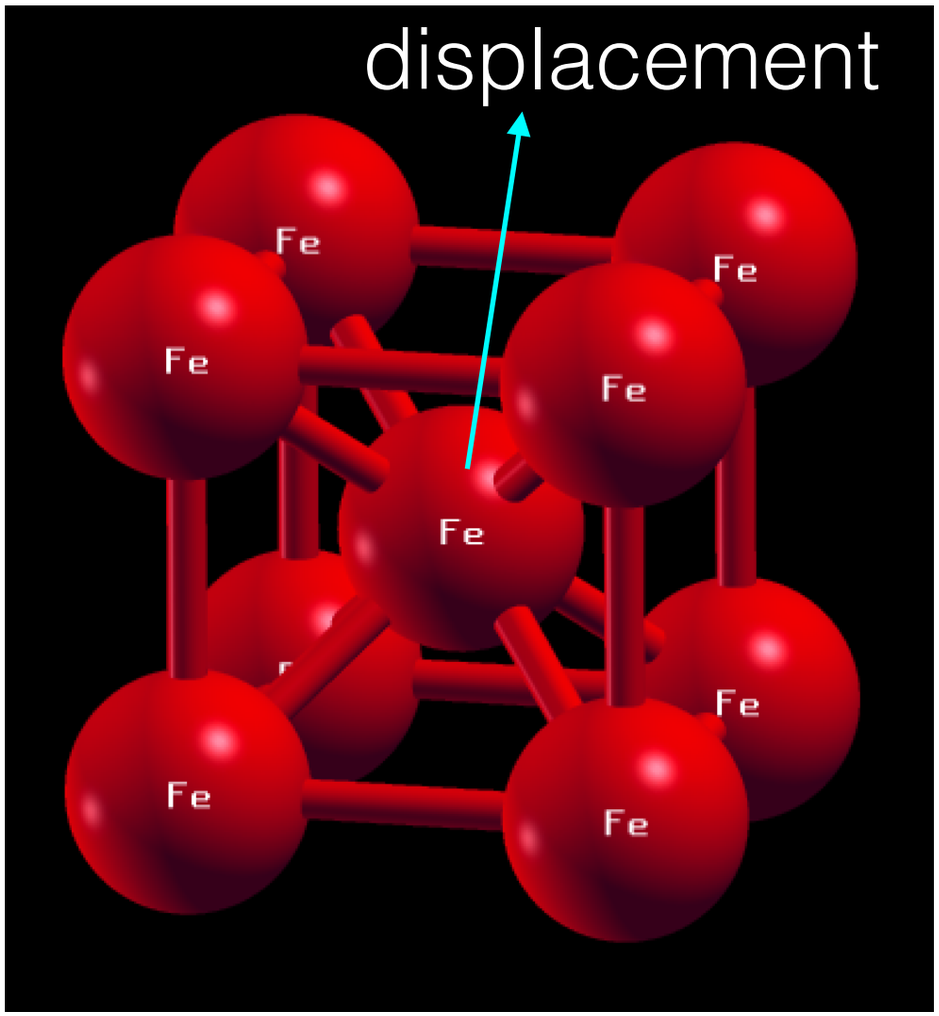


(Spin up channel)

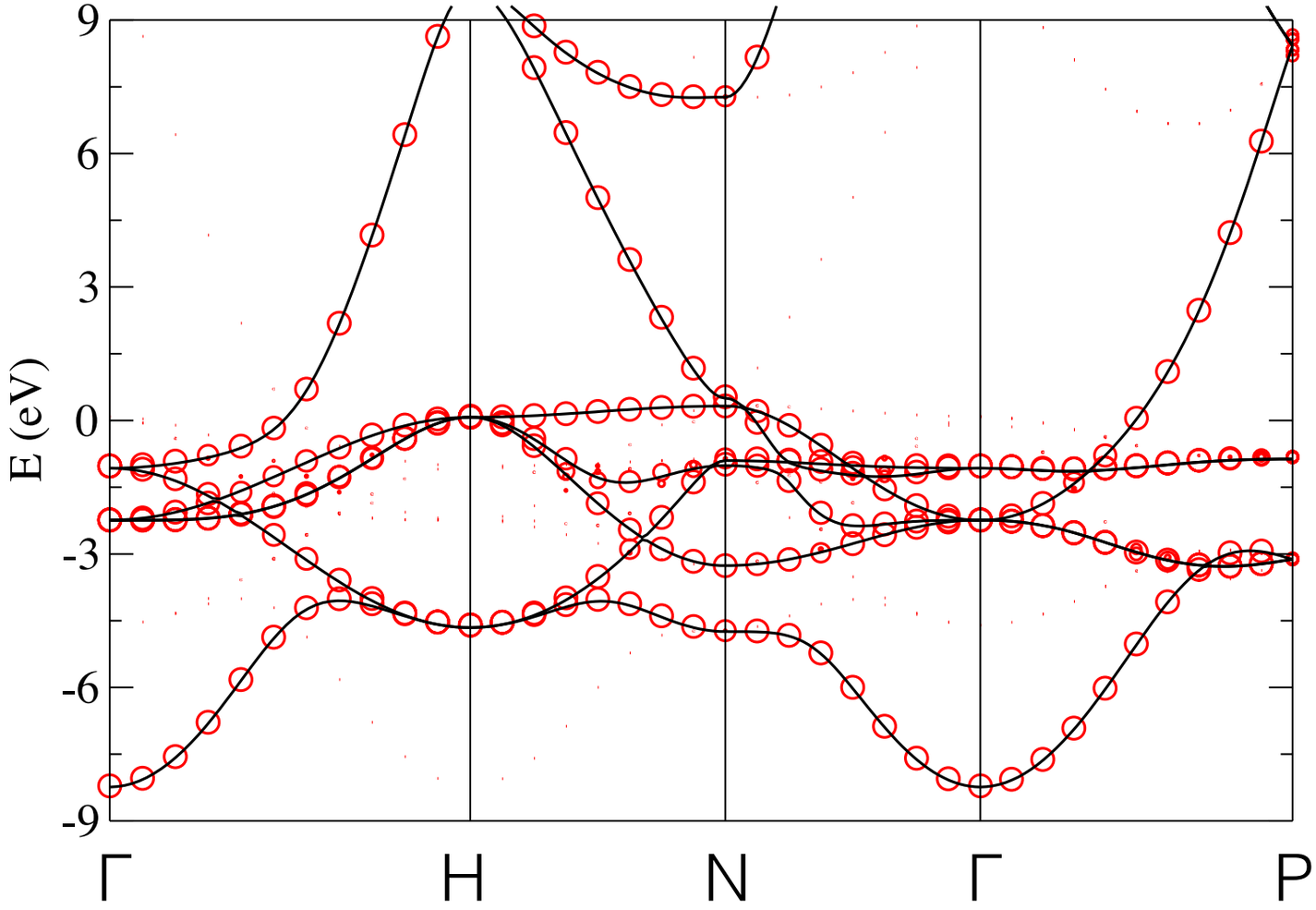
black curves are from primitive cell calculation

bcc Fe

unfolding is performed



conventional unit cell (two Fe atoms)



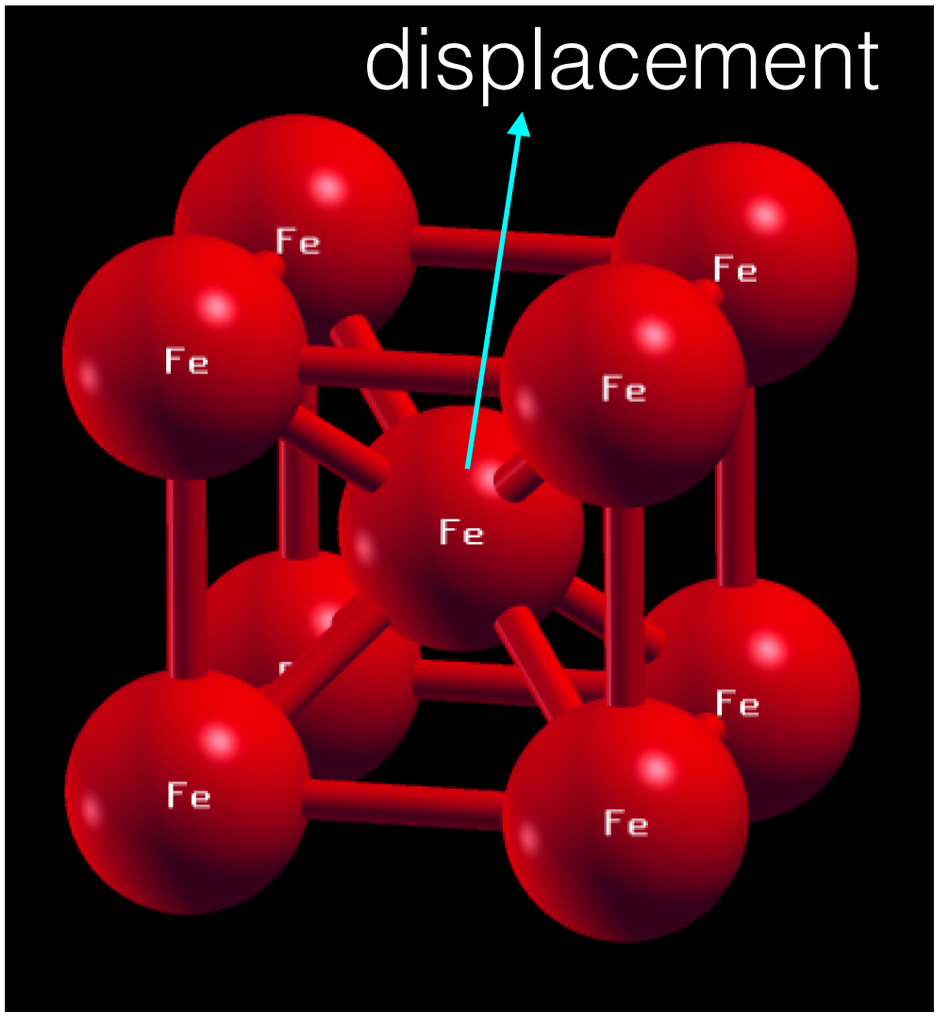

```
1   Fe   0.000   0.000   0.000
2   Fe   0.510   0.510   0.490

lattice constant: 2.87 Angstrom
```

(Spin up channel)

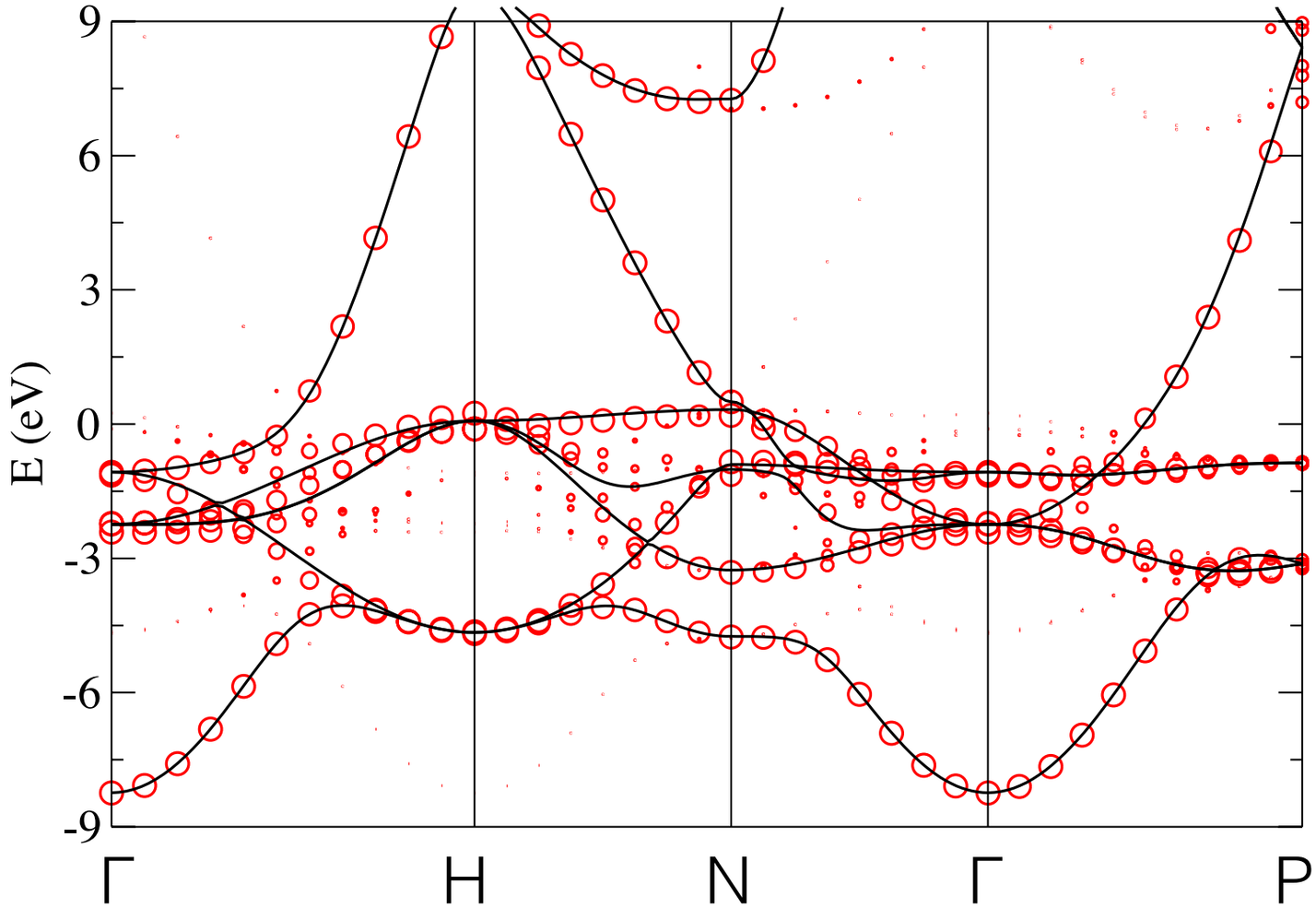
phonon effect is tiny on band structure

bcc Fe

unfolding is performed



conventional unit cell (two Fe atoms)




```
1  Fe  0.000  0.000  0.000
2  Fe  0.551  0.546  0.465

lattice constant: 2.87 Angstrom
```

(Spin up channel)

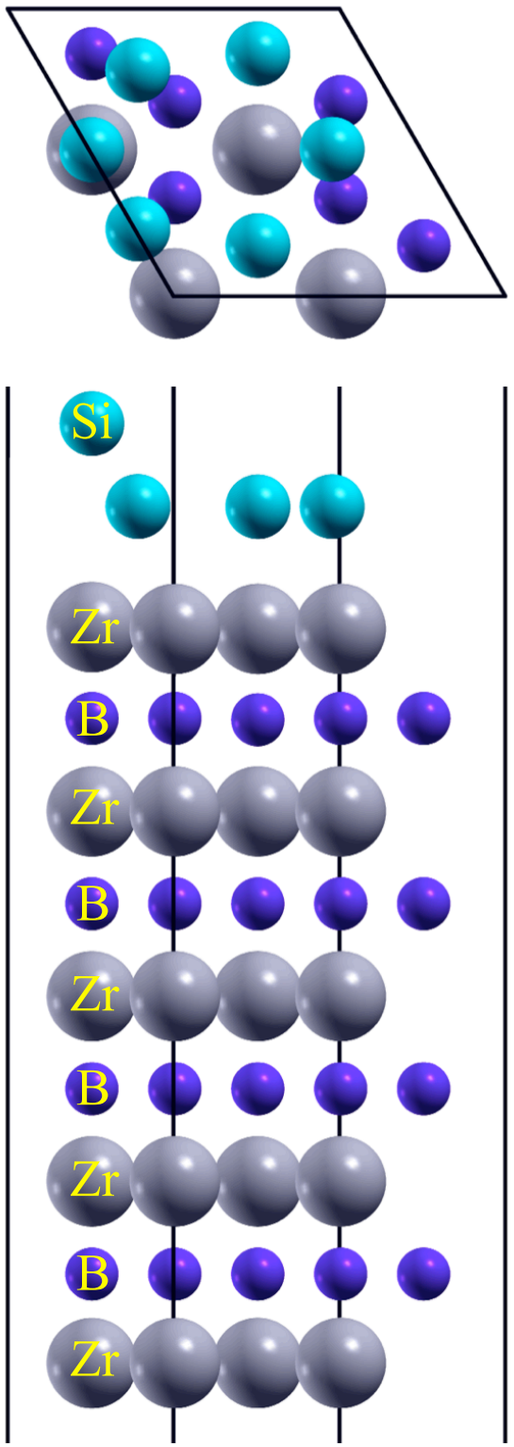
stronger translation symmetry breaking

The reference cell is conceptual

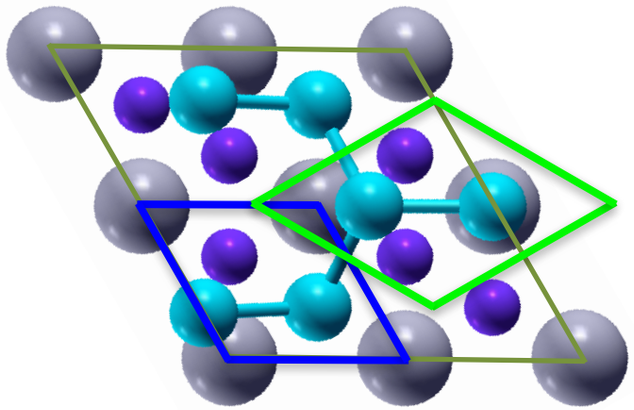
Silicene

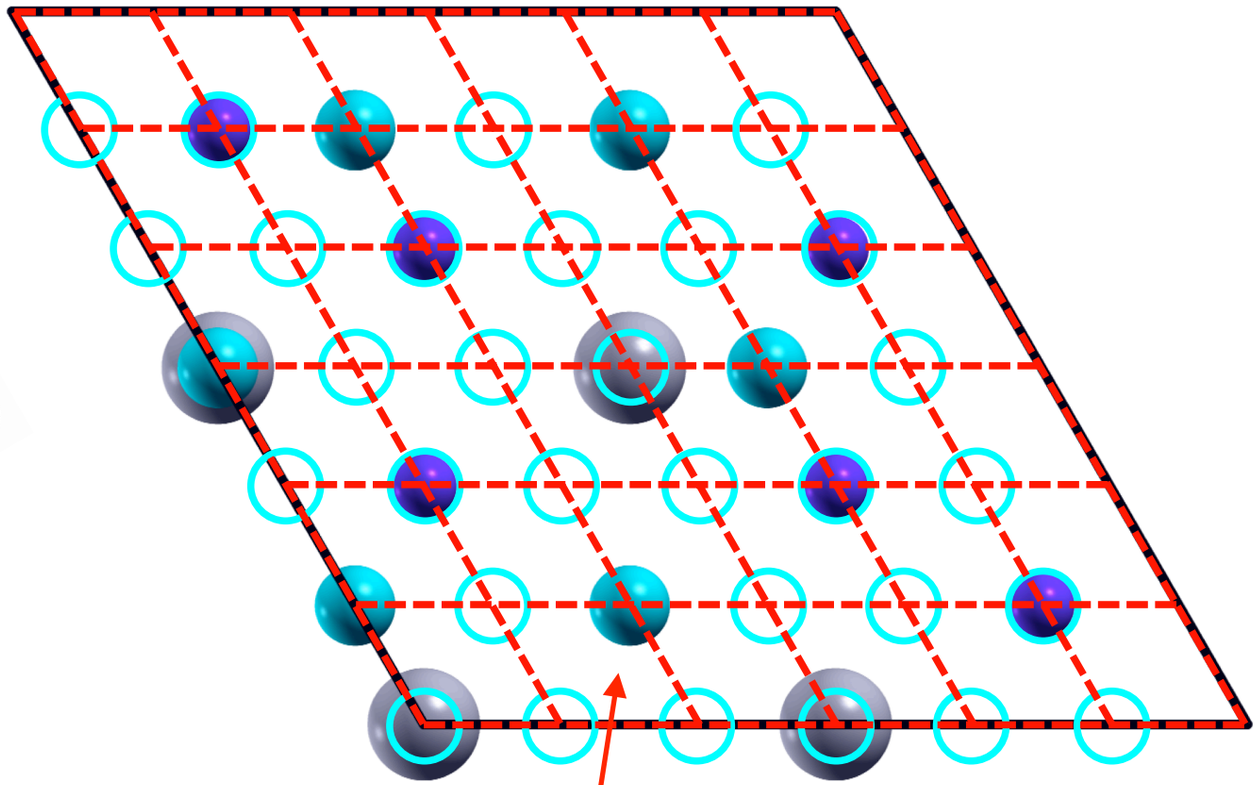
Silicene on ZrB_2 with incommensurate primitive cells (six Si atoms per unit cell)

Planar-like phase



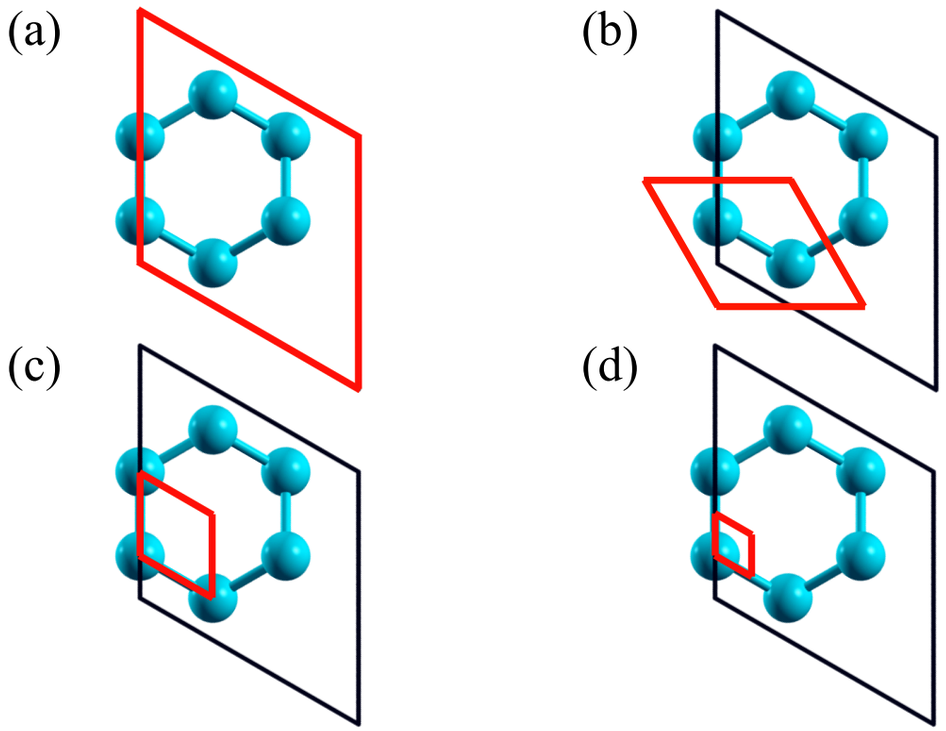
$\sqrt{3} \times \sqrt{3}$- reconstructed silicene





Silicene

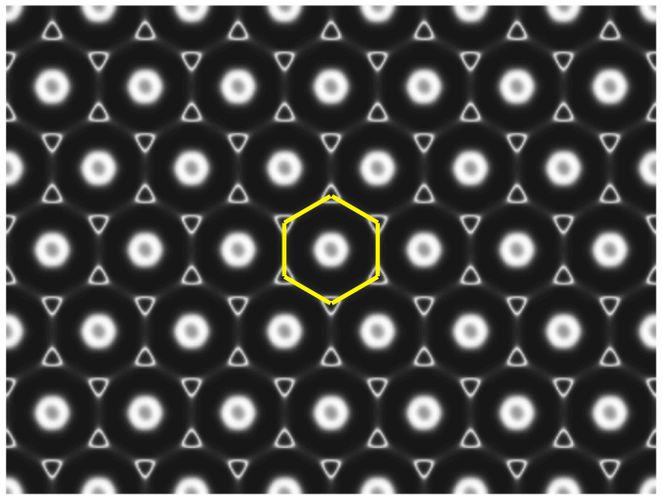
Free-standing planar-like silicene



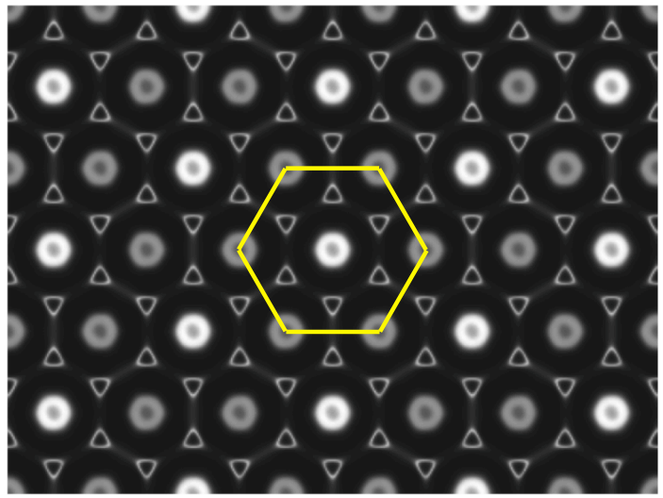
six Si atoms per unit cell is needed in the supercell

E = 1eV below the Fermi energy

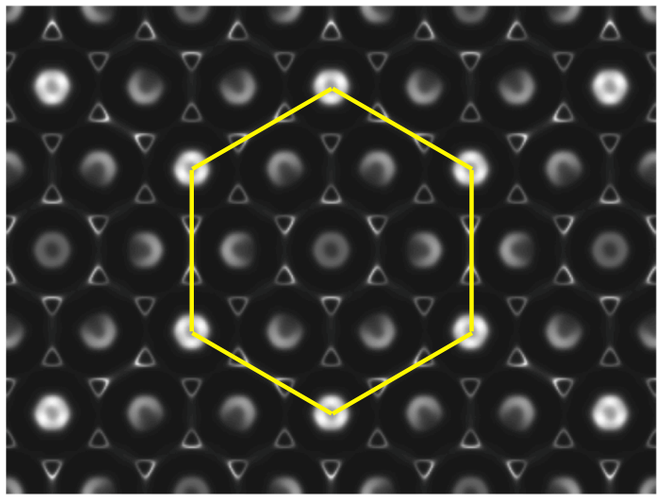
(i) Planar-like phase



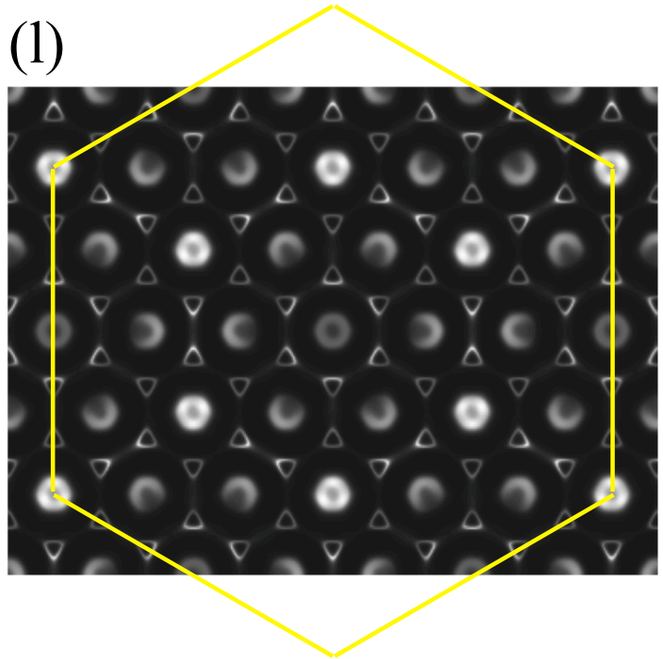
(j)



(k)



(l)

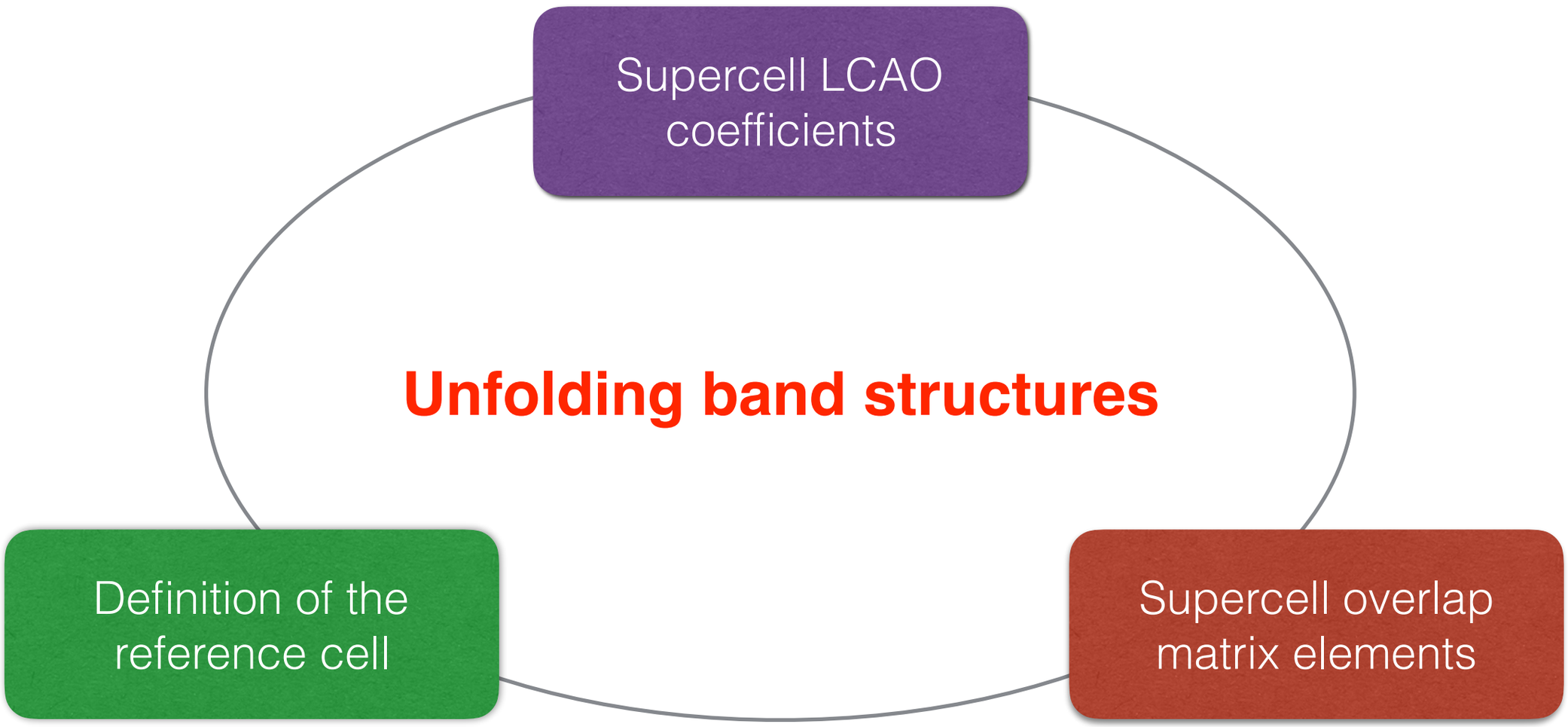


Chi-Cheng Lee *et al.*, Phys. Rev. B **90**, 075422 (2014).

Codes

Coding

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



Coding (currently working on source3.744)

Input file

```
Unfolding.fileout on
```
Turn on unfolding.

```
<Unfolding.ReferenceVectors
   1.4350   1.4350  -1.4350
  -1.4350   1.4350   1.4350
   1.4350  -1.4350   1.4350
Unfolding.ReferenceVectors>
```
Define reference lattice vectors.
The form and unit are the same as those in Atoms.UnitVectors.

```
<Unfolding.Referenceorigin
-0.1 -0.2 -0.3
Unfolding.Referenceorigin>
```
Define the origin of the reference unit cell. Together with the lattice vectors defined above, the reference cell is uniquely defined.

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

```
Unfolding.LowerBound -9.0
Unfolding.UpperBound  9.0
```
Define the energy window to perform the unfolding (in eV, to E_F)

```
Unfolding.desired_totalnkpt 40
```
Provide the desired total number of *k* points along the paths.

```
Unfolding.Nkpoint 5
```
Provide the number of high symmetry *k* points

```
<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>
```
List the high symmetry *k* points

```
<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;
```

Coding (currently working on source3.744)

<Input_std.c>

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.

```
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
```

Coding (currently working on source3.744)

<DFT.c>

```
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>**

```
      Band_DFT_MO
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);
  }
}
```

Coding (currently working on source3.744)

<Band_Unfolding.c>

```
void buildMapRlist();

double***** Elem;

int** Rlist;
```

size of Rlist is (#R,3)

Supercell overlap matrix elements

(already exist in the calculation)

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

<Band_Unfolding.c>

Definition of the **reference** cell

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

```
void buildtabr4RN(const double* a,const double* b,const double* c,const double* origin,const int* mapN2n);
```

```
int*** tabr4RN;
```

assign each supercell RN with reference-cell r

```
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

```
size of rnmap is (#r,#atom,2)
rnmap(i,j,0)=iR
rnmap(i,j,1)=iAtom
```

```
int** rlist;
```

size of rlist is (#r,3)

Coding (currently working on source3.744)

<Band_Unfolding.c>

Supercell LCAO coefficients

<Band_DFT_M0.c>

```
/*************************************************
      Solve eigenvalue problem at each k-point
*************************************************/

for (kloop=0; kloop<nkpoint; kloop++){

  if (myid==Host_ID && 0<level_stdout) printf("kpoint=%i /%i\n",kloop+1,nkpoint);

  k1 = kpoint[kloop][1];
  k2 = kpoint[kloop][2];
  k3 = kpoint[kloop][3];
…
  eigenvalue and eigenvector: EIGEN[spin][j1] and C[spin][j1][i1]
…
}
```

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>
```

<Band_Unfolding.c>

```
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)

<Band_Unfolding.c>

Supercell overlap matrix elements

Supercell LCAO coefficients

Definition of the reference cell

```
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
…
}
```

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^k_{KJM} as follows:

$$W^k_{KJM} = \frac{L}{l}\sum_G \delta_{k-G,K} C^{KJ}_M \sum_{rN} \mathrm{e}^{\mathrm{i}k(r-r'(M))} \times C^{KJ*}_N S_{0N,rm(M)}. \quad (26)$$

Then, the spectral function can be written as

$$A_{kj,kj}(\omega) = \sum_K A_{KJ,KJ}(\omega) \sum_M W^k_{KJM}. \quad (27)$$

Chi-Cheng Lee, Yukiko Yamada-Takamura, and Taisuke Ozaki
J. Phys.: Condens. Matter **25**, 345501 (2013).

Coding (currently working on source3.744)

<Band_Unfolding.c> efficiency can be improved later

```
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]);

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;
                r[0]=rlist[ir][0]*a[0]+rlist[ir][1]*b[0]+rlist[ir][2]*c[0];
                r[1]=rlist[ir][0]*a[1]+rlist[ir][1]*b[1]+rlist[ir][2]*c[1];
                r[2]=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));
}}}}
```

$$W^{k}_{KJM} = \frac{L}{l}\sum_{G}\delta_{k-G,K}C^{KJ}_{M}\sum_{rN}e^{ik(r-r'(M))} \times C^{KJ*}_{N}S_{0N,rm(M)}\cdot$$

rnmap

```
output contribution of each orbital for each |kj>: weight[Atom][Orbital]
or total weight:
double sumallorb=0.;
for (j=0; j<atomnum; j++) for (k=0; k<Norbperatom[j]; k++) sumallorb+=weight[j][k].r;
```

Coding (currently working on source3.744)

<Band_Unfolding.c> **we have two components in non-collinear case**

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

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

Coding (currently working on source3.744)

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
```

Coding (currently working on source3.744)

output file: case.out

collinear

non-collinear

```
**************************************************************
**************************************************************
                  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
Indiviual 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
         ...
```

Coding (currently working on source3.744)

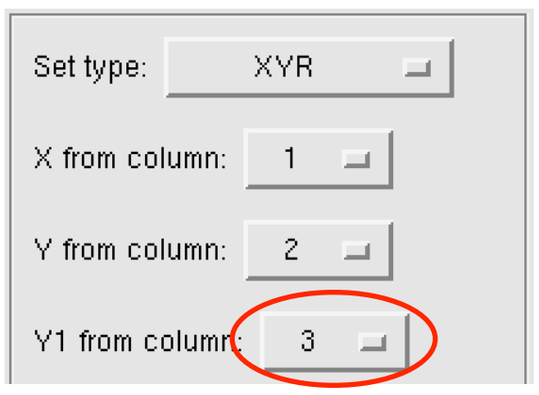
case.unfold_totup

```
0.000000 -8.207744  1.0000000
0.000000 -4.593939  0.0000000
0.000000 -4.593938  0.0000000
0.000000 -2.231539  1.0000000
0.000000 -2.231538  1.0000000
0.000000 -2.231536  1.0000000
0.000000 -1.012023  1.0000000
0.000000 -1.012021  1.0000000
0.000000 0.088024  0.0000000
0.000000 0.088025  0.0000000
0.000000 0.088027  0.0000000
0.105319 -8.041093  1.0000000
0.105319 -4.523466  0.0000000
...
```

case.unfold_orbup

```
0.000000 -8.207744 5.196924e-01 -2.133240e-02 1.640029e-03 -2.341670e-19 1.431638e-19 1.607270e-21 6.579302e-19 -9.507081e-20 7.292816e-22 5.011185e-19
1.495310e-19 -3.873358e-22 7.967105e-16 2.710925e-15 2.563743e-15 3.668372e-15 9.790445e-16 8.572274e-18 -4.608347e-17 1.251340e-16 2.020436e-16
2.646718e-17 5.196924e-01 -2.133240e-02 1.640029e-03 -1.463851e-19 2.098391e-19 -2.030736e-21 -4.225673e-20 -1.220893e-19 1.524443e-22 -4.667359e-19
1.707836e-19 4.982022e-22 7.823754e-16 2.587230e-15 2.526589e-15 3.750209e-15 9.692023e-16 9.663945e-18 -4.420333e-17 1.207162e-16 2.142916e-16
2.559349e-17
0.000000 -4.593939 8.548314e-17 -6.132439e-18 6.155895e-19 2.688336e-21 -2.071278e-21 -1.114958e-20 5.210173e-21 8.841696e-21 2.209782e-20 1.129659e-21
7.410371e-20 -1.575050e-21 4.772465e-10 -1.173963e-09 5.030595e-17 -4.240110e-18 2.239389e-18 -1.022076e-12 2.685972e-12 -1.755281e-18 3.269287e-18
1.112658e-20 -8.329702e-17 6.074079e-18 -5.370182e-19 2.032066e-21 -5.291468e-21 -1.456159e-20 -5.247778e-21 2.228774e-21 2.241918e-20 -1.151516e-21
-1.391200e-20 7.209190e-21 -4.772465e-10 1.173963e-09 -2.101113e-17 2.072213e-17 -2.164603e-18 1.022076e-12 -2.685972e-12 1.962646e-18 -3.204406e-18
-6.307238e-21
...
```

Grace



Gnuplot

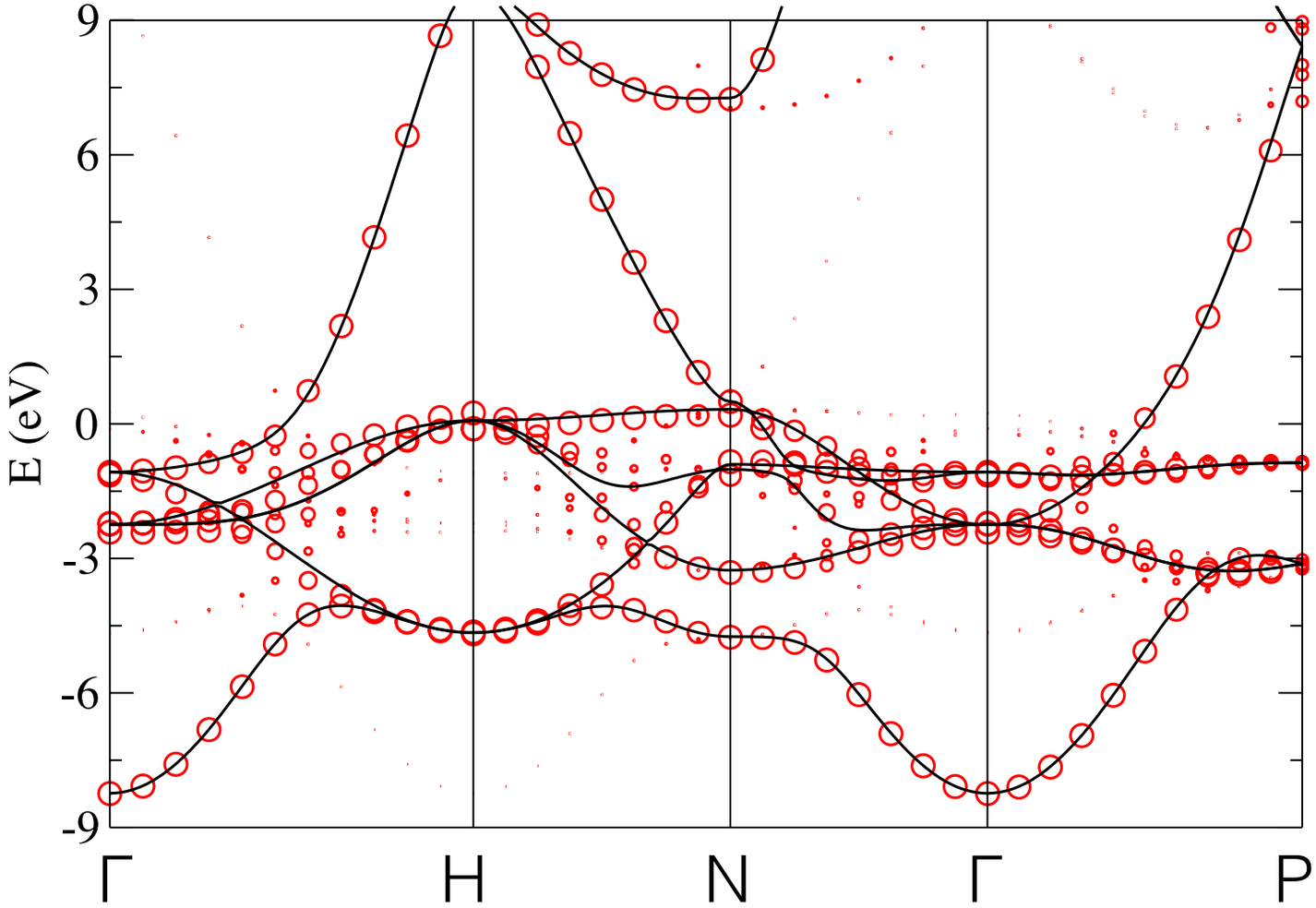
```
plot 'case.unfold_totup' using 1:2:($3)*0.05 with circles
```

be cautious with negative values in plotting the weight

Coding (currently working on source3.744)

Then you can, for example,

1. Prepare the figures together with the manuscript.



2. Cite J. Phys.: Condens. Matter **25**, 345501 (2013).
3. Submit to a journal.
4. Keep your fingers crossed.

Thank you for your attention!