PHYS 516: Methods of Computational Physics

ASSIGNMENT 6- TIGHT BINDING MODEL OF ELECTRONIC STRUCTURES

Anup V Kanale

March 22, 2017

1 The Hamiltonian Matrix

A programw as written to set up the Hamiltonian matrix and diagonalize it using the functions `tred2.c` and `tqli.c` from the book *Numerical Recipes*. The code is attached in the appendix, along with the plotting codes.

2 Effect of lattice constants on density of state

The density of states (DOS) of a system describes the number of states per interval of energy at each energy level that are available to be occupied. It is given by

$$D(\varepsilon) = \sum_{\nu=1}^{n4} \frac{1}{\sqrt{\pi}\sigma} \exp\left(\frac{-(\varepsilon - \varepsilon_\nu)^2}{\sigma^2}\right) \tag{1}$$

where $\sigma = 0.1eV$ is the energy spread given to each energy eigenvalue, ε_ν. The following parameters were used: `InitUcell[0] = InitUcell[1] = InitUcell[2] = 1`, `nAtoms = 8` and `LCNS = 1.8 × 5.43 A, 1.4 × 5.43 A, 1 × 5.43 A`, k_BT `= 0.2 eV`.

The plots below show the Density of States for different lattice constant values. We see that when the lattice contant is the largest (`LCNS = 1.8 × 5.43 A`), there are two sharp peaks, which means that when the atoms are furthest from each other, the density is high in the energy states E_s and $E - p$ and zero everywhere else.

When LCNS reduces slightly, the atoms are closer together and the density in the states in between increases.

For the case with the least LCNS ((`LCNS = 1.0 × 5.43 A`)), the atoms are very close together so the energy states in between also start getting populated.

3 Effect of number of atoms on density of state

The Density of states with a different number of periodic unit cells, i.e., a different number of atoms was plotted as shown in the figure below, for `LCNS = 1.0 × 5.43 A`. We see that the density of states has a larger spread now as more atoms interact with each other.

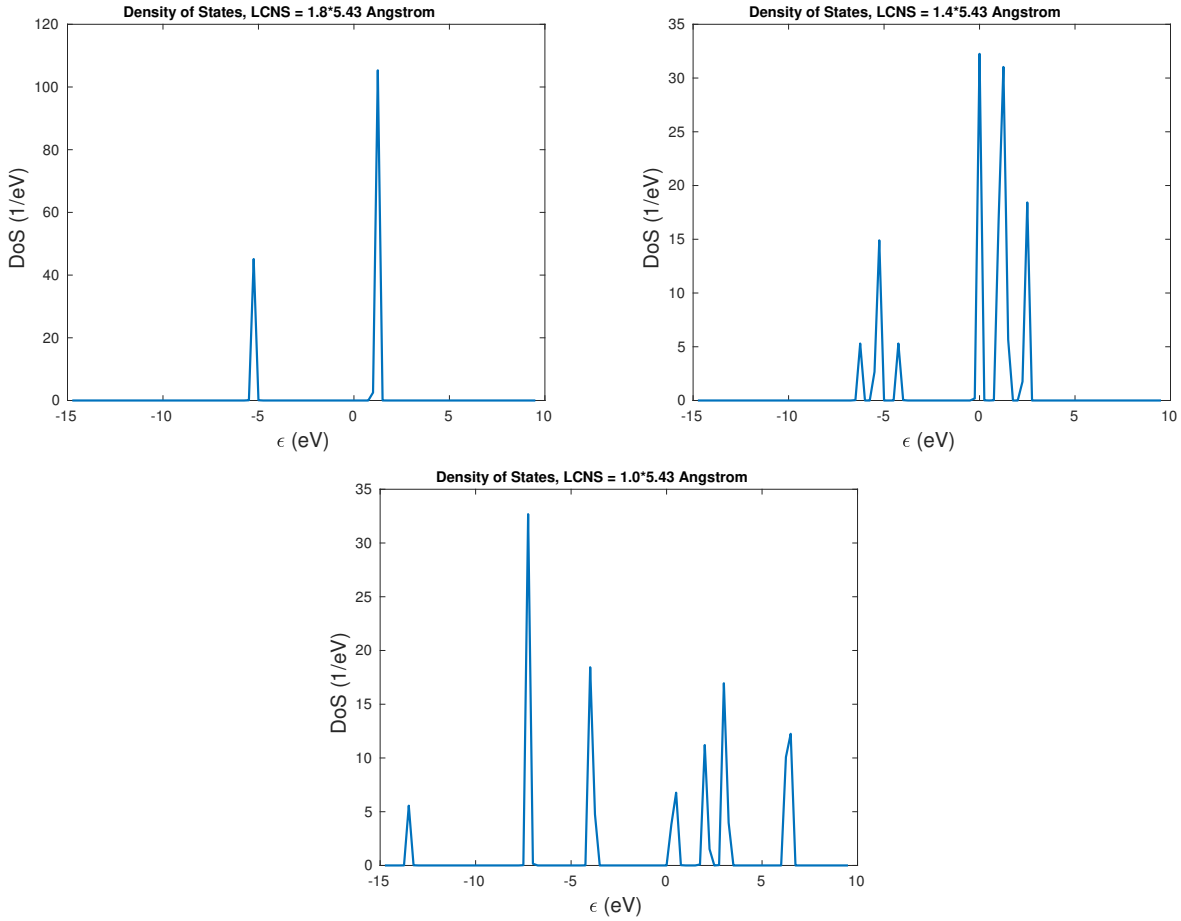


Figure 1: Density of States for 8 atoms

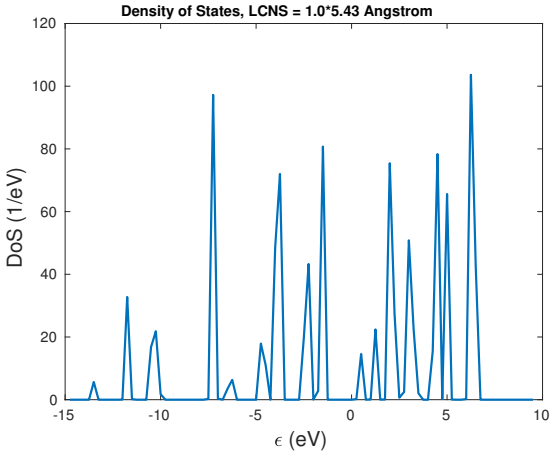


Figure 2: Density of States for 64 atoms

4 Fermi Distribution

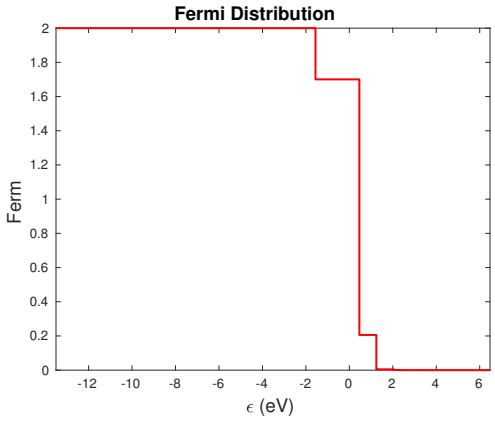
Fermi distribution represents the probablity of a state of energy being occupied by an electron. It is given by

$$f(\varepsilon_\nu) = \frac{2}{\exp\left\{\frac{\varepsilon_\nu - \mu}{k_B T}\right\} + 1} \tag{2}$$

I calculate the Fermi energy by first finding the chemical potential μ using Newton-Raphson method, using the fact that the integral of the distribution over energy gives the total number of electrons. Mathetically,

$$\sum_\nu f(\varepsilon_\nu) = 4N \tag{3}$$

The parameters used for the simulation were: `InitUcell[0] = InitUcell[1] = InitUcell[2] = 2`, `nAtoms = 64`, `LCNS = 5.43` A, k_BT `= 0.2 eV` and `eigenenergies = 256`. The plot below shows the distribution



A C Program

```
1  /* Program to simulate the Tight Binding Model of Electronic
       Structures */
2
3  #include <stdio.h>
4  #include <math.h>
5  #include <stdlib.h>
6
7  #define NMAX 100              /* Max # of atoms */
8  #define NAUC 8                /* # of atoms per unit cell */
9  #define LCNS 1.0*5.43         /* Lattice constant of Si (5.43 angstrom)
       in atomic unit */
10
11 int nAtom;                    /* # of atoms */
12 double r[NMAX][3];            /* r[i][0/1/2] is the x/y/z coordinate of
       atom i */
13 int InitUcell[3];             /* # of unit cells */
14 double RegionH[3];
15
16 double **dmatrix(int, int, int, int);
17 double *dvector(int, int);
18 void InitConf();
19 double SignR(double, double);
20 void tred2(double **, int, double *, double *);
21 void tqli(double *, double *, int, double **);
22 void computeDOS(double *, int);
23 void fermDist(double *, int);
24 void sortd(double *, int);
25
26 int main() {
27   InitConf();
28
29   double **h; // Hamiltonian matrix
30   double *d;  // Eigenenergies
31   double *e;  // Work array for matrix diagonalization
32   FILE *fp;
33
34   int n4;
35   n4 = 4*nAtom;
36   h = dmatrix(1,n4,1,n4);
37   d = dvector(1,n4);
38   e = dvector(1,n4);
39
40   int ii,jj,kk;
41   // Initialize Hamiltonian matrix
42   for (ii=1; ii<=n4; ii++) {
```

```
43      for (jj=1; jj<=n4; jj++) {
44        h[ii][jj] = 0;
45      }
46    }
47
48    /* Populate the Hamiltonian matrix */
49    double r0 = 2.360352, n = 2, Es = -5.25, Ep = 1.2; // Constants
50
51    // index— ss\sigma sp\sigma pp\sigma pp\pi
52    double hLambda_r0[] = {-2.038, 1.745, 2.75, -1.075};
53    double nLambda[] = {9.5, 8.5, 7.5, 7.5};
54    double rLambda[] = {3.4, 3.55, 3.7, 3.7};
55
56    double hLambda[] = {0,0,0,0};
57    double rx,ry,rz,rMag,dx,dy,dz;
58    double rij[] = {0,0,0};
59    double expterm;
60
61    for (jj=0; jj<nAtom; jj++) {
62      for (ii=0; ii<nAtom; ii++) {
63        if (ii==jj) {
64        // Diagonal elements
65        h[1+4*jj][1+4*ii] = Es;
66        h[2+4*jj][2+4*ii] = Ep;
67        h[3+4*jj][3+4*ii] = Ep;
68        h[4+4*jj][4+4*ii] = Ep;
69
70        // Off-diagonal elements are zero
71        }
72
73        else {
74        // Calc rij with min image convention
75        for (kk=0; kk<3; kk++) {
76          rij[kk] = r[jj][kk] - r[ii][kk];
77          /* Chooses the nearest image */
78          rij[kk] = rij[kk] - SignR(RegionH[kk],rij[kk]-RegionH[kk]) -
                SignR(RegionH[kk],rij[kk]+RegionH[kk]);
79        }
80
81        rx = rij[0];
82        ry = rij[1];
83        rz = rij[2];
84        rMag = sqrt(rx*rx + ry*ry + rz*rz);
85
86        dx = rx/rMag; dy = ry/rMag; dz = rz/rMag; // Unit vector
87
88        expterm = n*(-pow((rMag/rLambda[0]), nLambda[0]) +
              pow((r0/rLambda[0]),nLambda[0]));
```

```
89          hLambda[0] = hLambda_r0[0]* pow((r0/rMag),n)* exp(expterm); //
                h_ss\sigma
90
91          expterm = n*(-pow((rMag/rLambda[1]), nLambda[1]) +
                pow((r0/rLambda[1]),nLambda[1]));
92          hLambda[1] = hLambda_r0[1]* pow((r0/rMag),n)* exp(expterm); //
                h_sp\sigma
93
94          expterm = n*(-pow((rMag/rLambda[2]), nLambda[2]) +
                pow((r0/rLambda[2]),nLambda[2]));
95          hLambda[2] = hLambda_r0[2]* pow((r0/rMag),n)* exp(expterm); //
                h_ss\sigma
96
97          expterm = n*(-pow((rMag/rLambda[3]), nLambda[3]) +
                pow((r0/rLambda[3]),nLambda[3]));
98          hLambda[3] = hLambda_r0[3]* pow((r0/rMag),n)* exp(expterm); //
                h_pp\pi
99
100         // Diagonal elements
101         h[1+4*jj][1+4*ii] = hLambda[0];
102         h[2+4*jj][2+4*ii] = dx*dx*hLambda[2] + (1-dx*dx)*hLambda[3];
103         h[3+4*jj][3+4*ii] = dy*dy*hLambda[2] + (1-dy*dy)*hLambda[3];
104         h[4+4*jj][4+4*ii] = dz*dz*hLambda[2] + (1-dz*dz)*hLambda[3];
105
106         // Populate off-Diagonal elements
107         h[1+4*jj][2+4*ii] = dx*hLambda[1]; h[2+4*jj][1+4*ii] =
                -h[1+4*jj][2+4*ii];
108         h[1+4*jj][3+4*ii] = dy*hLambda[1]; h[3+4*jj][1+4*ii] =
                -h[1+4*jj][3+4*ii];
109         h[1+4*jj][4+4*ii] = dz*hLambda[1]; h[4+4*jj][1+4*ii] =
                -h[1+4*jj][4+4*ii];
110
111         h[2+4*jj][3+4*ii] = dx*dy*(hLambda[2] - hLambda[3]);
                h[3+4*jj][2+4*ii] = h[2+4*jj][3+4*ii];
112         h[2+4*jj][4+4*ii] = dx*dz*(hLambda[2] - hLambda[3]);
                h[4+4*jj][2+4*ii] = h[2+4*jj][4+4*ii];
113
114         h[3+4*jj][4+4*ii] = dy*dz*(hLambda[2] - hLambda[3]);
                h[4+4*jj][3+4*ii] = h[3+4*jj][4+4*ii];
115         }
116       }
117   }
118
119   // Print Hamiltonian matrix (to check diagonal dominance)
120   fp = fopen("hamiltonian.txt", "w");
121   fprintf(fp, "Hamiltonian matrix before diagonalizing\n");
122   fprintf(fp, "/*----------------------------------------*/\n");
123   for (ii=1; ii<=n4; ii++) {
```

```
124       for (jj=1; jj<=n4; jj++) {
125         fprintf(fp, "%12le  ", h[ii][jj]);
126       }
127       fprintf(fp, "\n");
128     }
129     fclose(fp);
130
131     /* Diagonalize the Hamiltonian matrix using Numerical Recipes
          functions*/
132     tred2(h,n4,d,e);
133     tqli(d,e,n4,h);
134
135     computeDOS(d, n4);  // Compute Density of States
136     sortd(d, n4);
137     fermDist(d, n4);  // Get Fermi distribution
138 }
139
140
141
142 void fermDist(double *d, int n4) {
143     /* Computes chemical potential mu using Newton Raphson method
144        and then computes the Fermi distribution and writes to file*/
145
146     FILE *fp;
147     double ferm[n4];
148     double expo, kbt = 0.2;
149     double fSum, dFdu, Fu;
150     double uNew, uTol=100.0;
151     double u = 1.8;
152     int input;
153
154     // Newton Raphson root finding to calculate mu
155     while(uTol>1e-5) {
156       fSum = 0.0;
157       dFdu = 0.0;
158       for (int ii=1; ii<=n4; ii++) {
159         expo = exp( (d[ii]-u)/kbt );
160         ferm[ii-1] = 2.0/(expo + 1);
161         fSum = fSum + ferm[ii-1];
162         dFdu = dFdu + 2.0/kbt*expo/pow((expo+1), 2);
163       }
164       Fu = fSum-n4;
165       uNew = u - Fu/dFdu;
166       uTol = fabs(uNew-u);
167       u = uNew;
168     }
169     printf("Converged value of mu is %le \n", uNew);
170
```

```
171   // Print Fermi distribution to file
172   fp = fopen("fermiDist.txt", "w");
173   for (int ii=1; ii<=n4; ii++) {
174     fprintf(fp, "%le \t %le \n", d[ii], ferm[ii]);
175   }
176   fclose(fp);
177 }
178
179
180 void sortd(double *d, int n4){
181   /* Sorts the eigen energy array in ascending order */
182
183   double dummy;
184   for (int ii=1; ii<=n4; ii++) {
185     for (int jj=ii+1; jj<=n4; jj++) {
186       if (d[ii]>d[jj]) {
187         dummy = d[ii];
188         d[ii] = d[jj];
189         d[jj] = dummy;
190       }
191     }
192   }
193 }
194
195 /*-------------------------------------------------------------------------------*/
196 void computeDOS(double *d, int n4){
197   /* Compute Density of States given the eigenvalues in vector d
198      Vary totStates for finer resolution */
199
200   FILE *fp;
201   int ii,kk,totStates = 100;
202   double sigma = 0.1, deps;
203   deps= (double) 25/totStates;
204   double Dens[totStates], eps[totStates];
205
206   // discretize Eigen energies
207   for (ii=0; ii<totStates-1; ii++) {
208     if (ii==0) {eps[0] = -15.0;}
209     else {eps[ii] = eps[ii-1] + deps;}
210   }
211
212   // Calculate DOS for each eigen energy
213   for (ii=1; ii<totStates; ii++) {
214     Dens[ii] = 0;
215     for (kk=1; kk<=n4; kk++) {
216       Dens[ii] = Dens[ii] + 1/(sqrt(M_PI)*sigma) * exp(-pow( (
              (eps[ii] - d[kk])/sigma),2 ));
217     }
```

```
218    }
219
220    // Write to file
221    fp = fopen("DensOStates.txt", "w");
222    for (ii=1; ii<totStates-1; ii++) {
223      fprintf(fp, "%le \t %le \n", eps[ii], Dens[ii]);
224    }
225    fclose(fp);
226  }
227  /*----------------------------------------------------------------------*/
228
229
230
231  /*----------------------------------------------------------------------*/
232  double SignR(double v,double x) {
233    /* Applies minimum image convention to find closest neighbour
234       in the presense of periodic boundary conditions */
235
236    if (x > 0) return v;
237    else return -v;
238    }
239  /*----------------------------------------------------------------------*/
240
241
242
243  void InitConf() {
244  /*----------------------------------------------------------------------
245    r are initialized to diamond lattice positions.
246  ----------------------------------------------------------------------*/
247    double gap[3];       /* Unit cell size */
248    double c[3];
249    int j,k,nX,nY,nZ;
250    /* Atom positions in a unit diamond crystalline unit cell */
251    double origAtom[NAUC][3] = {{0.0, 0.0, 0.0 }, {0.0, 0.5, 0.5 },
252                                {0.5, 0.0, 0.5 }, {0.5, 0.5, 0.0 },
253                                {0.25,0.25,0.25}, {0.25,0.75,0.75},
254                                {0.75,0.25,0.75}, {0.75,0.75,0.25}};
255
256    /* Read the # of unit cells in the x, y & z directions */
257    scanf("%d%d%d",&InitUcell[0],&InitUcell[1],&InitUcell[2]);
258
259    for (k=0; k<3; k++){
260      RegionH[k] = 0.5*LCNS*InitUcell[k];
261    }
262
263    /* Sets up a diamond lattice */
264    for (k=0; k<3; k++) gap[k] = LCNS;
265    nAtom = 0;
```

```
266   for (nZ=0; nZ<InitUcell[2]; nZ++) {
267     c[2] = nZ*gap[2];
268     for (nY=0; nY<InitUcell[1]; nY++) {
269       c[1] = nY*gap[1];
270       for (nX=0; nX<InitUcell[0]; nX++) {
271         c[0] = nX*gap[0];
272         for (j=0; j<NAUC; j++) {
273           for (k=0; k<3; k++)
274             r[nAtom][k] = c[k] + gap[k]*origAtom[j][k];
275           ++nAtom;
276         }
277       }
278     }
279   }
280 }
```

siCrystal.c

```
1  close all; clear all;
2
3 fs = 14; % Font Size
4 %% Scatter Plots for enerygy
5 %--------------------------------------------------
6 energyData = dlmread('DensOStates.txt');
7 eps = energyData(:,1);
8 dos = energyData(:,2);
9
10 figure()
11 plot(eps, dos, 'LineWidth', 1.5);
12 xlabel('\epsilon (eV)','FontSize', fs); ylabel('DoS
      (1/eV)','FontSize', fs);
13 title('Density of States, LCNS = 1.0*5.43 Angstrom', 'FontSize', fs-4);
14
15 energyData = dlmread('fermiDist.txt');
16 d = energyData(:,1);
17 ferm = energyData(:,2);
18
19 figure()
20 plot(d, ferm, 'r-', 'LineWidth', 1.5);
21 xlim([d(1) d(end)])
22 xlabel('\epsilon (eV)','FontSize', fs); ylabel('Ferm','FontSize', fs);
23 title('Fermi Distribution', 'FontSize', fs);
```

siCrystal_Plot.m