Energy Summation in NaCl Crystal

Mehrdad Yousefi

March 15, 2017

Abstract

In this report, I described the detailed methodology of energy summation for NaCl crystal in real space lattice. Also Madelung constants are calculated for different situations when we have only Coulomb and Van der Waals types potentials and also when we have compensation charge modifcation in Coulomb law. Results show that because the Madelung constant for Coulomb part of potential energy is a Reimann series, we don't have absolute convergence but when we consider compensation charge term in potential equation, it forces the convergence for Coulomb Madelung constant.

Computational methods

In this section first of all, a brief description of written code in C++ will be discussed and then the results will be provided.

C++ codes to calculate the energy of NaCl crystal

In order to find the energy distribution in NaCl crystal, we should consider two mechanisms. First dominant mechanism in the ionic NaCl crystal, is electrostatic charge potential which is described by Coulomb's law. Also we should consider a term in energy functional to demonstrate the very weak psuedo Van der Waals interaction. These two mentioned factors are shown in Eq. (1).

$$\Phi_{ij} = \frac{\pm e^2}{r_{ij}} + \frac{\alpha}{r_{ij}^n} \tag{1}$$

Also if we define the dimensionless distance as $\rho_{ij} = \frac{r_{ij}}{r}$, we could rewrite the Eq. (1) as follows (Eq. (2)):

$$\Phi_{ij} = \frac{-e^2}{r} \sum_{hkl=-\infty}^{+\infty} \frac{(-1)^{h+k+l}}{(h^2+k^2+l^2)^{1/2}} + \frac{\alpha}{r^n} \sum_{hkl=-\infty}^{+\infty} \frac{1}{(h^2+k^2+l^2)^{n/2}} \tag{2}$$

So now, we can define the first and second Madelung constants as follows (Eq. (3) and Eq. (4)):

$$M_1 = \sum_{hkl=-\infty}^{+\infty} \frac{(-1)^{h+k+l}}{(h^2+k^2+l^2)^{1/2}} \tag{3}$$

$$M_2 = \sum_{hkl=-\infty}^{+\infty} \frac{1}{(h^2+k^2+l^2)^{n/2}} \tag{4}$$

So inorder to find these Madelung constants and energy functional numerically, this piece of code is written in C++ language and serial mode (Listing 1).

Listing 1: Unmodified Madelung Constants

```
#include <stdio.h>
#include <stdlib.h> // dynamic allocation
#include <math.h>
#include <iostream>
#include <vector>
#include <cassert>
#include <fstream>
#include <cmath>
#include <string>

using namespace std;

const unsigned int nx=200;
const unsigned int ny=200;
const unsigned int nz=200;
const unsigned int n=8.1;
```

```
const double r0=2.82;
const double epsilon=1e-1;

double Madelung_constant1() {

int h,k,l;
double sum;
sum=0.0;

 for (h=0; h<nx+1; h++) {
  for (k=0; k<ny+1; k++) {
   for (l=0; l<nz+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += pow(-1.0,h+k+l)/sqrt(pow(h,2)+pow(k,2)+pow(l,2));
 }


 }
 }
 }

 return sum;

}

void Madelung_constant1_convergence(FILE* stream) {

int h,k,l;
double sum;
sum=0.0;

for (int x=3; x<201; x++) {

 for (h=0; h<x+1; h++) {
  for (k=0; k<x+1; k++) {
   for (l=0; l<x+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += pow(-1.0,h+k+l)/sqrt(pow(h,2)+pow(k,2)+pow(l,2));
 }


 }
 }
 }

 fprintf(stream, "%d\t%f\n", x, sum);

 sum=0.0;
```

```
 }

}

double Madelung_constant2() {

int h,k,l;
double sum;
sum=0.0;

 for (h=0; h<nx+1; h++) {
  for (k=0; k<ny+1; k++) {
   for (l=0; l<nz+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += 1.0/sqrt(pow(pow(h,2)+pow(k,2)+pow(l,2),n));
 }


 }
 }
 }

 return sum;

}

double Madelung_constant2_convergence(FILE* stream) {

int h,k,l;
double sum;
sum=0.0;

for (int x=3; x<201; x++) {

 for (h=0; h<x+1; h++) {
  for (k=0; k<x+1; k++) {
   for (l=0; l<x+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += 1.0/sqrt(pow(pow(h,2)+pow(k,2)+pow(l,2),n));
 }


 }
 }
 }

 fprintf(stream, "%d\t%f\n", x, sum);
```

```
 sum=0.0;

 }

}

void Energy(vector<vector<vector<double>>> &U,
            double alpha, double M1, double M2) {

double r;

for (int i=0; i<=nx; i++) {
for (int j=0; j<=ny; j++) {
for (int k=0; k<=nz; k++) {

if (i==0 && j==0 && k==0) {
r = r0*epsilon;
} else {
r = r0*sqrt(pow(i,2)+pow(j,2)+pow(k,2));
}

U[i][j][k]=((-M1)/r)+(alpha*M2)/pow(r,n);

}
}
}

}

void write_output_vtk(vector<vector<vector<double>>>& U,
                      int nx, int ny, int nz)
{
    string name = "./output.vtk";
    ofstream ofile (name);

    // vtk preamble
    ofile << "# vtk DataFile Version 2.0" << endl;
    ofile << "OUTPUT by LIBM\n";
    ofile << "ASCII" << endl;

    // write grid
    ofile << "DATASET RECTILINEAR_GRID" << endl;
    ofile << "DIMENSIONS " << nx << " " << ny << " " << nz << endl;
    ofile << "X_COORDINATES " << nx << " float" << endl;
    for(size_t i = 0; i < nx; i++)
        ofile << i << "\t";
    ofile << endl;
    ofile << "Y_COORDINATES " << ny << " float" << endl;
    for(size_t i = 0; i < ny; i++)
```

```
        ofile << i << "\t";
    ofile << endl;
    ofile << "Z_COORDINATES " << nz << " float" << endl;
    for(size_t i = 0; i < nz; i++)
        ofile << i << "\t";
    ofile << endl;

    // point data
    ofile << "POINT_DATA " << nx*ny*nz << endl;

    // write rho
    ofile << "SCALARS " << "U" << " double" << endl;
    ofile << "LOOKUP_TABLE default" << endl;
  for (int k = 0; k < nz; k++)
    for(int j = 0; j < ny; j++)
        for(int i = 0; i < nx; i++)
            ofile << U[i][j][k] << endl;


}


int main(int argc, char** argv) {

FILE* f_convergence;
char filename[40];
double M1, M2;
double alpha = 684;
vector<vector<vector<double>>> U
(nx+1,vector<vector<double>>(ny+1,vector<double>(nz+1,0)));

M1 = Madelung_constant1();

M2 = Madelung_constant2();

sprintf(filename, "./Madelung1.txt");

f_convergence = fopen(filename, "w");

Madelung_constant1_convergence(f_convergence);

fclose(f_convergence);

sprintf(filename, "./Madelung2.txt");

f_convergence = fopen(filename, "w");

Madelung_constant2_convergence(f_convergence);

fclose(f_convergence);
```

```
Energy(U, alpha ,M1,M2);

write_output_vtk(U, nx+1,ny+1,nz+1);

printf("The first Madelung constant is = %f
       and the second one is = %f\n",M1,M2);

}
```

After running this code, we extract the convergence results and potential energy distribution in one and three dimensional plots (Fig. 1, Fig. 2, Fig. 3 and Fig. 4).

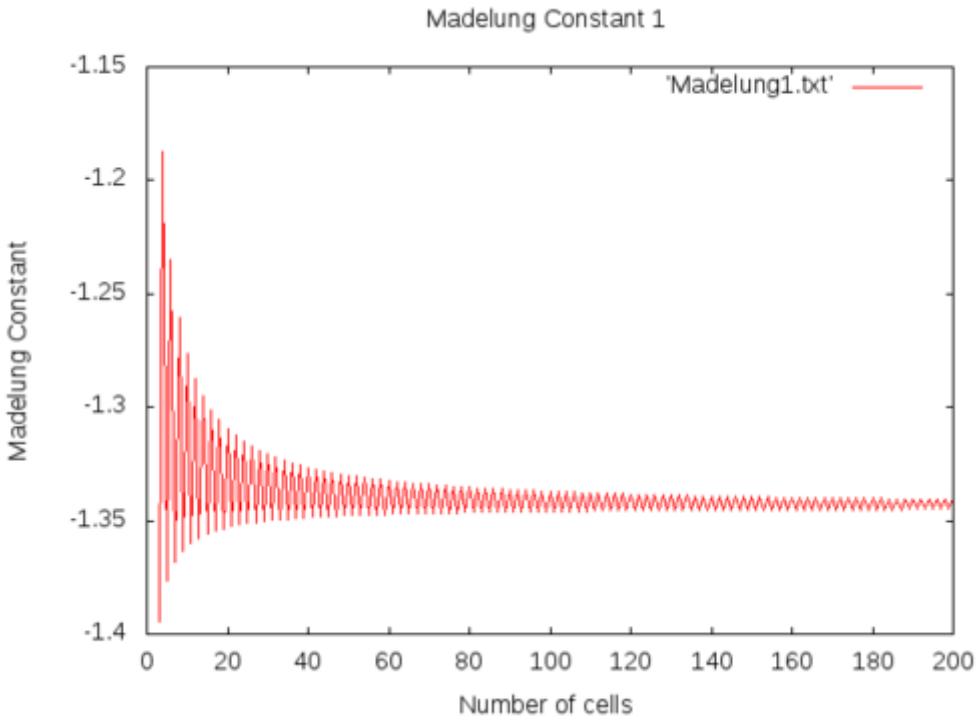

Figure 1: First Madelung constant convergence versus number of cells.

According to Fig. 1, we can find out that as I mentioned before the first Madelung constant which is related to Coulomb interaction, because of its long range nature, there is no absolute convergence for Eq. (1). But as you can see, when we increase the number of cells, it just bounce around a tiny range. So we could say that it just has a conditional convergence which is the main behavior of Reimann series. On the other hand, as you can see in Fig. 2, because of short range nature of pseudo Van der Waals potential, second Madelung constant converges very fast and doesn't show bouncing around a specific value. Also in Fig. 3 and Fig. 4, you can see the three and one dimensional energy distribution functional respectively.

In order to modify and enhance the conditional convergence behavior of Coulomb energy potential, we could introduce charge compensation term in the energy functional as follows (Eq. (5)).

$$\Phi_{ij} = \frac{\pm e^2}{r_{ij} e^{\beta r_{ij}}} + \frac{\alpha}{r_{ij}^n} \tag{5}$$

So if modify the Listing 1 to cover the compensation term, we could write this Listing 2 as follows:

Listing 2: Modified Madelung Constants

```
#include <stdio.h>
#include <stdlib.h> // dynamic allocation
#include <math.h>
#include <iostream>
#include <vector>
#include <cassert>
#include <fstream>
#include <cmath>
#include <string>

using namespace std;

const unsigned int nx=200;
const unsigned int ny=200;
const unsigned int nz=200;
const unsigned int n=8.1;
const double r0=2.82;
const double epsilon=1e-1;
const double beta=0.5;

double Madelung_constant1() {

int h,k,l;
double sum;
sum=0.0;

 for (h=0; h<nx+1; h++) {
  for (k=0; k<ny+1; k++) {
   for (l=0; l<nz+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += pow(-1.0,h+k+l)/(sqrt(pow(h,2)+pow(k,2)+pow(l,2))*
                          exp(beta*r0*sqrt(pow(h,2)+pow(k,2)+pow(l,2))));
 }


 }
 }
 }
```

```
 return sum;

}

void Madelung_constant1_convergence1(FILE* stream) {

int h,k,l;
double sum;
sum=0.0;

for (int x=3; x<201; x++) {

 for (h=0; h<x+1; h++) {
  for (k=0; k<x+1; k++) {
   for (l=0; l<x+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += pow(-1.0,h+k+l)/(sqrt(pow(h,2)+pow(k,2)+pow(l,2))*
                          exp(beta*r0*sqrt(pow(h,2)+pow(k,2)+pow(l,2))));
 }


 }
 }
 }

 fprintf(stream, "%d\t%f\n", x, sum);

 sum=0.0;

 }


}

void Madelung_constant1_convergence2(FILE* stream) {

int h,k,l;
double sum;
sum=0.0;

for (double betap=0.0; betap<1.1; betap += 0.01) {

 for (h=0; h<nx+1; h++) {
  for (k=0; k<ny+1; k++) {
   for (l=0; l<nz+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += pow(-1.0,h+k+l)/(sqrt(pow(h,2)+pow(k,2)+pow(l,2))*
```

```
                                exp(betap*r0*sqrt(pow(h,2)+pow(k,2)+pow(l,2))));
 }


 }
 }
 }

 fprintf(stream, "%f\t%f\n", betap, sum);

 sum=0.0;

 }


}

double Madelung_constant2() {

int h,k,l;
double sum;
sum=0.0;

 for (h=0; h<nx+1; h++) {
  for (k=0; k<ny+1; k++) {
   for (l=0; l<nz+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += 1.0/sqrt(pow(pow(h,2)+pow(k,2)+pow(l,2),n));
 }


 }
 }
 }

 return sum;

}

void Madelung_constant2_convergence(FILE* stream) {

int h,k,l;
double sum;
sum=0.0;

for (int x=3; x<201; x++) {

 for (h=0; h<x+1; h++) {
  for (k=0; k<x+1; k++) {
```

```
    for (l=0; l<x+1; l++) {

 if (h!=0 || k!=0 || l!=0) {
 sum += 1.0/sqrt(pow(pow(h,2)+pow(k,2)+pow(l,2),n));
 }


 }
 }
 }

 fprintf(stream, "%d\t%f\n", x, sum);

 sum=0.0;

 }

}

void Energy(vector<vector<vector<double>>> &U,
             double alpha, double M1, double M2) {

double r;

for (int i=0; i<=nx; i++) {
for (int j=0; j<=ny; j++) {
for (int k=0; k<=nz; k++) {

if (i==0 && j==0 && k==0) {
r = r0*epsilon;
} else {
r = r0*sqrt(pow(i,2)+pow(j,2)+pow(k,2));
}

U[i][j][k]=((-M1)/r)+(alpha*M2)/pow(r,n);

}
}
}

}

void write_output_vtk(vector<vector<vector<double>>>& U,
                      int nx, int ny, int nz)
{
    string name = "./output-modified.vtk";
    ofstream ofile (name);

    // vtk preamble
    ofile << "# vtk DataFile Version 2.0" << endl;
```

```
    ofile << "OUTPUT by LIBM\n";
    ofile << "ASCII" << endl;

    // write grid
    ofile << "DATASET RECTILINEAR_GRID" << endl;
    ofile << "DIMENSIONS " << nx << " " << ny << " " << nz << endl;
    ofile << "X_COORDINATES " << nx << " float" << endl;
    for(size_t i = 0; i < nx; i++)
        ofile << i << "\t";
    ofile << endl;
    ofile << "Y_COORDINATES " << ny << " float" << endl;
    for(size_t i = 0; i < ny; i++)
        ofile << i << "\t";
    ofile << endl;
    ofile << "Z_COORDINATES " << nz << " float" << endl;
    for(size_t i = 0; i < nz; i++)
        ofile << i << "\t";
    ofile << endl;

    // point data
    ofile << "POINT_DATA " << nx*ny*nz << endl;

    // write rho
    ofile << "SCALARS " << "U" << " double" << endl;
    ofile << "LOOKUP_TABLE default" << endl;
  for (int k = 0; k < nz; k++)
    for(int j = 0; j < ny; j++)
        for(int i = 0; i < nx; i++)
            ofile << U[i][j][k] << endl;


}


int main(int argc, char** argv) {

FILE* f_convergence;
char filename[40];
double M1, M2;
double alpha = 684;
vector<vector<vector<double>>> U
(nx+1,vector<vector<double>>(ny+1,vector<double>(nz+1,0)));

M1 = Madelung_constant1();

M2 = Madelung_constant2();

sprintf(filename, "./Madelung1_modified1.txt");

f_convergence = fopen(filename, "w");
```

```
Madelung_constant1_convergence1(f_convergence);

fclose(f_convergence);

sprintf(filename, "./Madelung1_modified2.txt");

f_convergence = fopen(filename, "w");

Madelung_constant1_convergence2(f_convergence);

fclose(f_convergence);

sprintf(filename, "./Madelung2_modified.txt");

f_convergence = fopen(filename, "w");

Madelung_constant2_convergence(f_convergence);

fclose(f_convergence);

Energy(U, alpha ,M1,M2);

write_output_vtk(U,nx+1,ny+1,nz+1);

printf("The first Madelung constant is = %f and
       the second one is = %f\n",M1,M2);

}
```

Now, we can extract those results which are shown in Fig. 1, Fig. 2, Fig. 3 and Fig. 4 for modified energy functional as follows (Fig. 5, Fig. 6 and Fig. 7).

As you can see in Fig. 5, by adding the compensation term, we modified convergence behavior of first Madelung series successfully and now it doesn't show conditional convergence and bouncing around a specific value. Also you can see the variation of converged first Madelung constant versus β parameter in Fig. 6. For better comparison, the one-dimensional energy functional for with and without compensation term is plotted in Fig. 7.

Three-dimensional energy distribution in NaCl XTAL

By doing summation all over a NaCl crystal, we could extract three-dimensional energy distribution. In order to find the desired functional, we could define the total energy of XTAL as follows in Eq. (6).

$$E(r) = \sum_{r_{ij} \subset XTAL} \Phi_{ij}(r) \tag{6}$$

So by calculating the summation in Eq. (6), we could extract the three and one dimensional energy distribution for a $20 \times 20 \times 20$ NaCl crystal which are shown in Fig. 8 and Fig. 9 respectively.

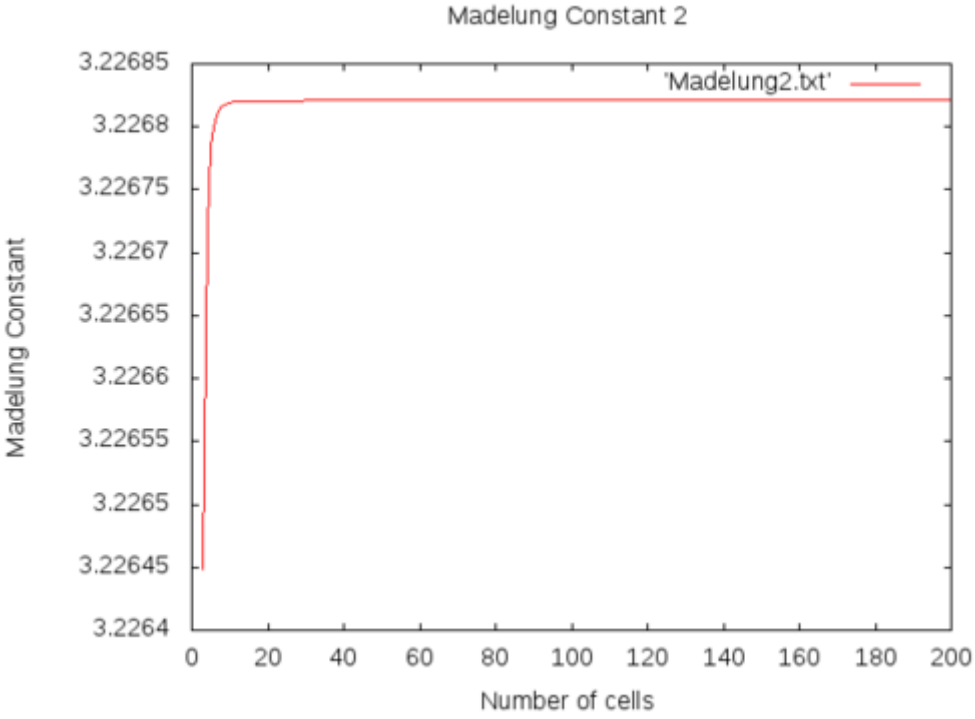

Figure 2: Second Madelung constant convergence versus number of cells.

Discussion

The results show that by considering proper energy functional we could extract the behavior of ionic XTAL precisely. Also in order to decrease the aggressive behavior of Coulomb potential, we should consider a compensation term to have a smooth convergence. This compensation term can modify the incorrect ignorance of valence shell electrons and force the convergence to the correct value.

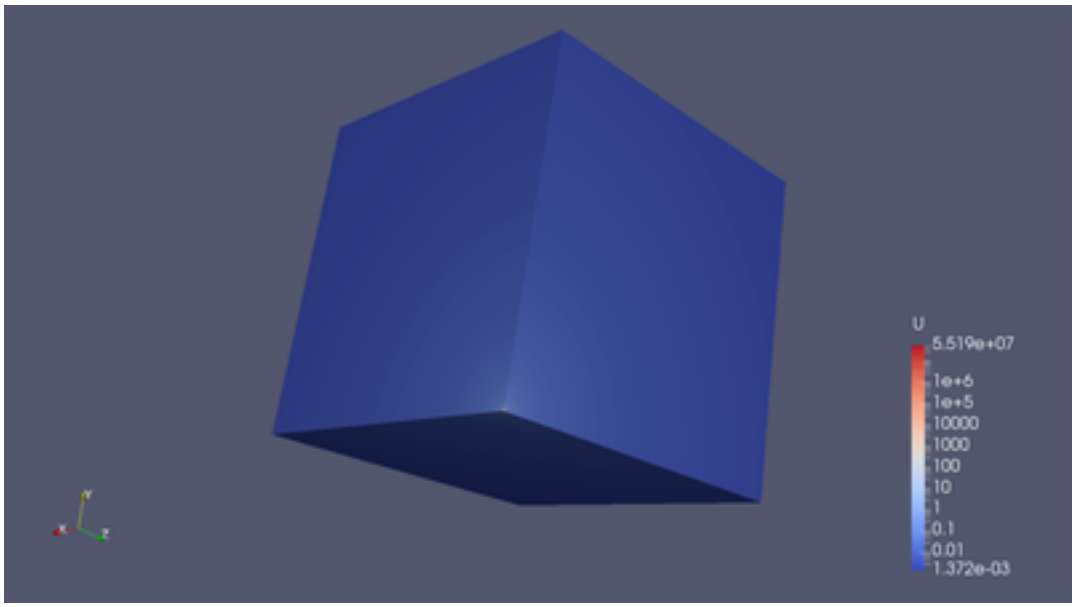

Figure 3: Three-dimensional energy distribution for Na atom.

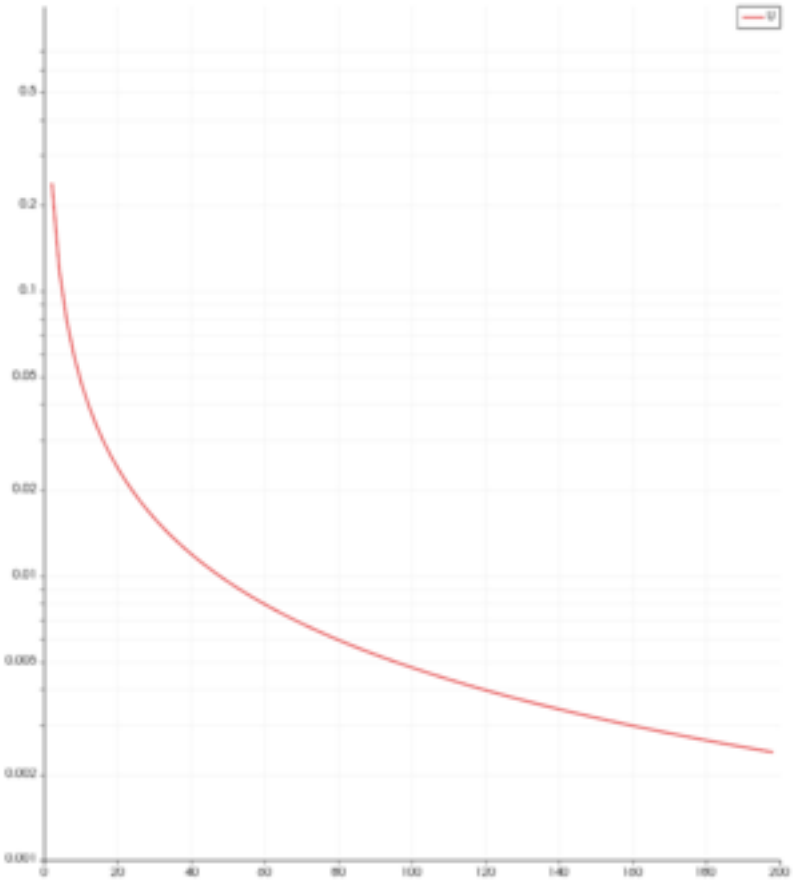

Figure 4: One-dimensional energy distribution for Na atom.

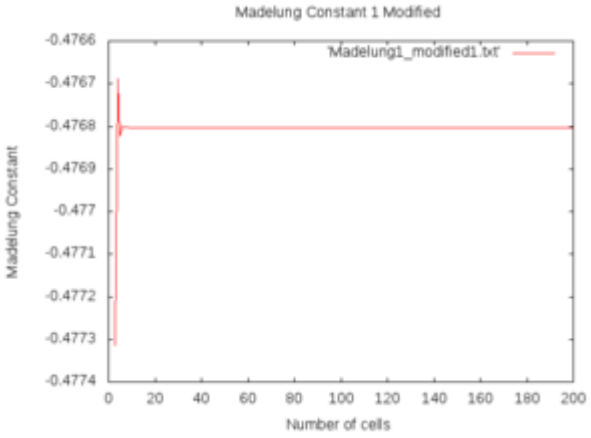

Figure 5: First Madelung constant convergence versus number of cells for modified energy functional.

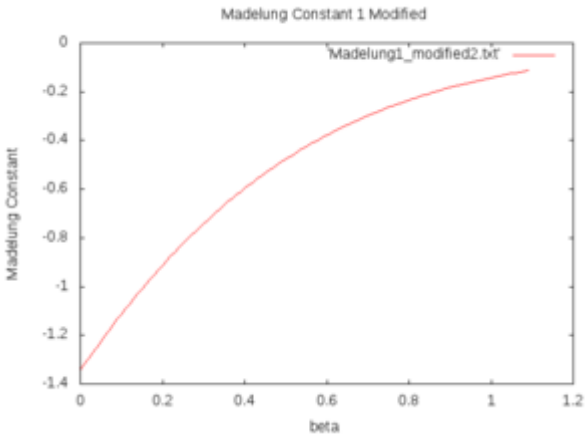

Figure 6: First Madelung constant convergence versus β parameter for modified energy functional

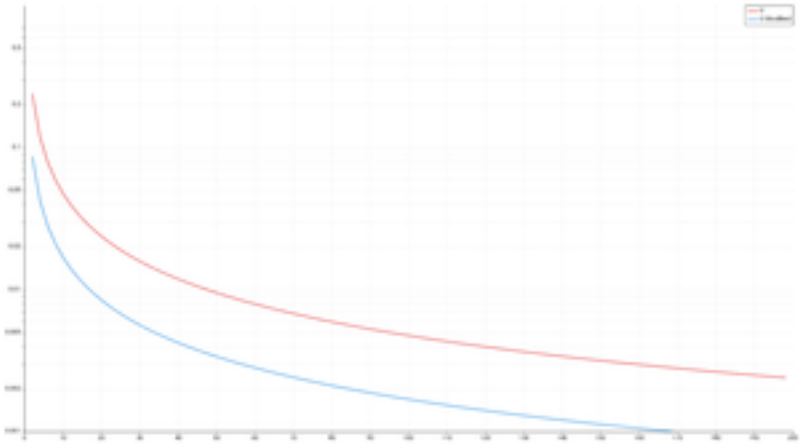

Figure 7: One-dimensional energy distribution for Na atom in with and without compensation term (red = without compensation and blue = with compensation).

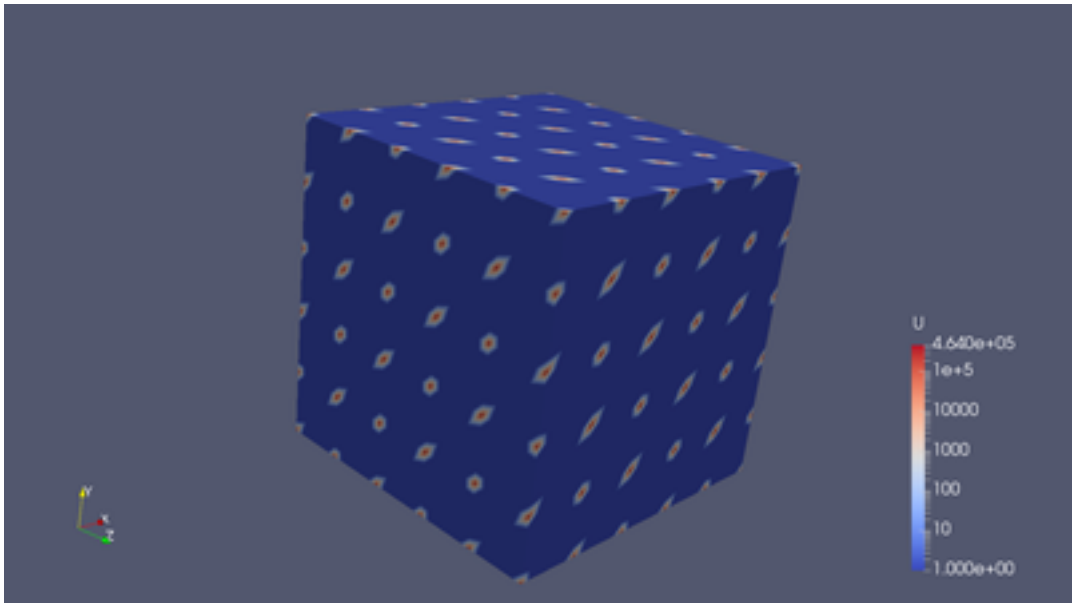

Figure 8: Three-dimensional energy distribution for NaCl XTAL.

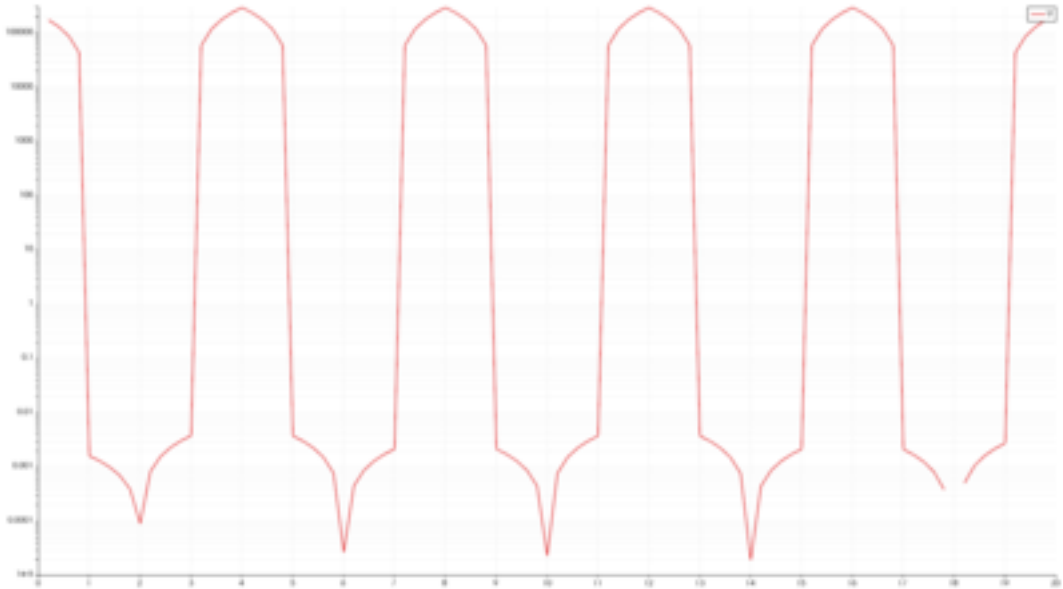

Figure 9: One-dimensional energy distribution for NaCl XTAL.