Density of States of Impure Palladium Deutride/Hydride

by

Irfan Ullah Chaudhary

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1996

Author .....
Department of Electrical Engineering and Computer Science
February, 1996

Certified by .....
Peter L. Hagelstein
Associate Professor
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Density of States of Impure Palladium Deutride/Hydride

by

Irfan Ullah Chaudhary

Submitted to the Department of Electrical Engineering and Computer Science
on February, 1996, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The dynamical matrix method with fixed force constants is used to calculate the density of states of a palladium deutride/hydride lattice with vacancies. Perturbative and non-perturbative techniques are used to try and speed up the computation. The most important computation is the density of states calculation of a perfect palladium deutride/hydride lattice with one Pd vacancy. It is found that the vacancy modes pile up near the bottom of the optical band.

Thesis Supervisor: Peter L. Hagelstein
Title: Associate Professor

Density of States of Impure Palladium Deutride/Hydride

by

Irfan Ullah Chaudhary

Submitted to the Department of Electrical Engineering and Computer Science
on February, 1996, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The dynamical matrix method with fixed force constants is used to calculate the density of states of a palladium deutride/hydride lattice with vacancies. Perturbative and non-perturbative techniques are used to try and speed up the computation. The most important computation is the density of states calculation of a perfect palladium deutride/hydride lattice with one Pd vacancy. It is found that the vacancy modes pile up near the bottom of the optical band.

Thesis Supervisor: Peter L. Hagelstein
Title: Associate Professor

Acknowledgments

MIT has been a bittersweet and strange experience. In retrospective, I think, the human experience (however little!!) has a much more lasting impression than the technical knowledge gained. Peter has been a source of inspiration. He is the perfect example of how an "ideal" professor should be in an ideal world (unfortunately the world is somewhat non-ideal!!). Owl and Bird have been great....Ammee and Abbu totally amazing. Without Idol's sarcasm, life wouldn't have been as much "fun". Asad helped me a lot with the technical stuff, and I think Sumanth's spirit was somehow trying to guide me to write bug-free code (although I don't think it succeeded!). He also wrote a part of the code for the Embedded Atom Method. Jim, (as he has been for the past 6 years) was a great help, friend and mentor. Anyway the list can continue, and I have to catch a flight tomorrow in the afternoon.....

"I'm out of here.....I'm history. No! I'm mythology. Nah! I don't care what I am. I am free!!!" The Genie in Aladdin.

Contents

1 Introduction **9**

1.1 Palladium 12

1.2 Palladium deutride/hydride 14

2 Theoretical Background **17**

2.1 The Dynamical Matrix 17

2.2 Building The Dynamical Matrix 19

2.3 Speeding up the calculation 20

2.3.1 Symmetries in the Brillouin zone 20

2.3.2 Non-degenerate Perturbation Theory 25

2.3.3 Degenerate Perturbation Theory 26

2.3.4 Linear Extrapolation 28

3 Results **30**

3.1 Checks 30

3.1.1 Phonon Spectrum 31

3.1.2 Density of states 31

3.2 Comparison Of Various Methods 32

3.3 Dilute Limit 35

3.4 The Fukai Structure 38

3.5 Vacancy modes near a Pd vacancy 38

4 Conclusion and Extensions **42**

4.1 Main results 42
4.2 Relaxation 44
4.2.1 Embedded Atom Method 44
4.2.2 Minimization 44
4.2.3 Results 45
4.3 Further Research 45

A Force Constants For Different Directions **47**

B Construction Of The Dynamical Matrix **49**

C Detailed Results Of Calculations **52**

D The Code **63**

List of Figures

1-1 The top figure shows the palladium lattice. The bottom figure shows the palladium hydride lattice. 13

1-2 The top figure shows a bcc lattice. The bottom figure shows the first Brillouin zone of Pd or PdH 15

2-1 The top figure shows the dispersion relation for a cell with lattice constant a_o. The bottom figure shows the same dispersion relation for a supercell with lattice constant $2a_o$. 21

2-2 The figure shows two Brillouin zones. The square BZ does not cause any problems when scaling down. However the bottom one causes boundary problems. 23

3-1 Density of state for different PdD supercells (in arbitrary units). Top left is $1 \times 1 \times 1$, top right is $2 \times 2 \times 2$, bottom left is $3 \times 3 \times 3$, bottom right is $4 \times 4 \times 4$ 33

3-2 The top two figures show the density of states for a $3 \times 3 \times 3$ cell with a deuterium atom added at $a_o(1, 1, 1/2)$. The bottom figures show the density of states when hydrogen atom is added at the same position. . 36

3-3 The top two figures show the density of states for a $3 \times 3 \times 3$ cell with a deuterium atom added at $a_o(1, 1, 1/2)$ and a Pd atom removed from $a_o(1, 1, 1)$. The bottom figures show the density of states for the same structure with hydrogen replacing the deuterium. 37

3-4 The figure shows the density of states for the Fukai structure for Palladium Deutride 39

3-5 The top figures show the density of states for a perfect Palladium Deutride lattice. The bottom figure shows the density of states with Pd at $a_o(1,1,1)$ missing 40

3-6 The top figures show the density of states for a perfect Palladium Hydride lattice. The bottom figure shows the density of states with Pd at $a_o(1,1,1)$ missing 41

B-1 A simple 2 dimensional structure. 49

C-1 Cell size is $1 \times 1 \times 1$. No perturbation theory or linear extrapolation is used. 53

C-2 Cell size is $1 \times 1 \times 1$. The graphs are calculated using non-degenerate perturbation theory 54

C-3 Cell size is $1 \times 1 \times 1$. First 3 graphs are calculated using degenerate perturbation theory. The last graph is calculated using non-degenerate perturbation theory with linear extrapolation 55

C-4 Cell size is $1 \times 1 \times 1$. The graphs are calculated using non-degenerate perturbation theory with linear extrapolation 56

C-5 Cell size is $1 \times 1 \times 1$. The first graph is calculated using non-degenerate perturbation theory with linear extrapolation. The last three graphs use degenerate perturbation theory with linear extrapolation 57

C-6 Cell size is $2 \times 2 \times 2$. Perturbation theory or linear extrapolation were not used for the calculation of these graphs 58

C-7 Cell size is $2 \times 2 \times 2$. The first graph is calculated using non-degenerate perturbation theory, the second with degenerate perturbation theory, and the last two with non-degenerate perturbation theory with linear extrapolation 59

C-8 Cell size is $3 \times 3 \times 3$. No Perturbation theory or linear extrapolation is used in the calculation of the first three graphs. The last graph uses non-degenerate perturbation theory with linear extrapolation 60

List of Tables

1.1 Relation between size of unit cell and time taken to diagonalize a $m \times m$ matrix(m is the number in column 2) 11

2.1 Force Constants in dyns/cm 18

3.1 Comparison of convergence times and iterations for density of states calculation using the root sampling method 32

3.2 Comparison of convergence times and iterations for density of states calculation using non-degenerate perturbation theory with linear extrapolation 32

3.3 comparison of convergence times for different methods 34

B.1 The toy force constants 50

C.1 Comparison of various methods 61

C.2 Mistakes for a $1 \times 1 \times 1$ cell 62

C.3 Mistakes for a $2 \times 2 \times 2$ cell 62

C.4 Mistakes for a $3 \times 3 \times 3$ cell 62

Chapter 1

Introduction

The calculation of the frequency spectrum of an impure lattice is an old problem. It was first tackled by Lifshitz in the 1940s[1]. He considered a single impurity atom in an otherwise perfect lattice [1]. Since then many, including Maradudin [3] and Dawber and Elliot [2] have considered similar such problems. The main results of their analysis were that:

- the perturbed modes with frequencies in the continuum range were changed near the defect atom
- when a light impurity atom was introduced into the lattice, localized modes with frequencies above the range of unperturbed modes appear.

We are specifically interested in a lattice of palladium deutride/hydride. A great deal of literature is available on this specific lattice [4, 5, 6, 7]. Unfortunately all the papers have only considered substituting hydrogen with vacancies or other atoms. This is no coincidence. The reason for such a choice is that when people are doing experiments they try to load palladium with hydrogen. Ideally PdD/H would have a $NaCl$ like structure.[2]. But hydrogen does not fill all the octahedral sites in the lattice, and hence it is physically interesting to consider the case of PdD_x or PdH_x, where x is any real number between 0 and 1.

[1] The force constants for an imperfect lattice are assumed to be the same as the ones for a perfect lattice

[2] See Figure 1-1 for the structure

However our interest is in a palladium deutride/hydride structure which has Pd vacancies. It is interesting physics in its own right, but we are mostly interested in this structure because it plays an important part in the Coherent Neutron Transfer Theory [8, 9] put forward by Peter Hagelstein. According to this theory there is a mechanism through which the lattice and the nucleus can exchange energy. This is a very startling claim, because solid state and nuclear physicists, both, claim that such an energy transfer is absolutely impossible. The reason is very simple: nuclear energy is on the order of MeVs whereas lattice energy is on the order of eVs. In order for a nucleus to couple to a lattice, 10^6 phonons need to be destroyed/created. The probability of such an occurrence is vanishingly small.

But there *is* another way for a lattice to transfer energy. Let us assume that there is a large number of phonons,N, in one mode. If through some mechanism, these phonon modes are shifted by $\delta\omega$. Then there is a net change in energy of

$$\Delta E = N\hbar\delta\omega$$

This energy is transfered to the mechanism which caused the change in the frequency in the first place.

Palladium is much heavier as compared to deuterium/hydrogen.[3] Thus it oscillates at a significantly lower frequency as compared to the deuterium/hydrogen. The low frequency region is called the "acoustical band" and the high frequency region corresponding to the oscillating deuterium/hydrogen atom is labeled as the "optical band". However if we make a palladium vacancy, then the 6 deuterium/hydrogen atoms surrounding it will vibrate at a lower frequency. And if the frequency is such that it falls in between the optical and the acoustical modes then the impurity modes are said to be in the band gap. If this were the case then the 18 phonon modes would drop down from the optical band to the vacancy impurity band. The excess energy ΔE would go into the nuclear process which caused the Pd vacancy to be created in the first place. So our hope is that when we create a Pd vacancy in the palladium

[3] about 50/100 times

deutride/hydride lattice then we will find the phonon modes in the band gap.

The way we propose to solve our problem is theoretically quite straight forward. We take a $n \times n \times n$ super cell of palladium hydride/deutride. From this supercell we remove one or more Pd or D/H atoms. Then using periodic boundary conditions, we set up the dynamical matrix of this supercell, and diagonalize this matrix to get the eigenvalues. The numerical details are rather tiresome, but the theory is very well established. The details can be found in the next chapter and in Appendix B.

But even before doing any calculations, it is obvious that the size of the matrix and the diagonalization time increases very rapidly with n, the size of the supercell. This can be seen from Table 1.1.[4]

size of unit cell	size of matrix	t_1	t_2
1	6	< 1	< 1
2	48	< 1	1
3	162	7	32
4	384	87	415

Table 1.1: Relation between size of unit cell and time taken to diagonalize a $m \times m$ matrix(m is the number in column 2)

To get the density of states calculation to converge, we needed to form and diagonalize this dynamical matrix at approximately 500 points in the Brillouin zone. Thus the problem becomes rather unmanageable for a $3 \times 3 \times 3$ or a $4 \times 4 \times 4$ supercell. Various methods were tried to speed up this computation. They are discussed in Chapter 2. In the next two sections we discuss the structure of the direct and reciprocal lattices of Pd and PdD/H.

[4] t_1 in Table 1.1 refers to the time it takes the LAPACK routine to diagonalize a hermitian matrix without calculating the eigenvectors, and t_2 refers to the same diagonalization with calculation of eigenvectors

1.1 Palladium

As shown in Figure 1-1 palladium is a fcc lattice. It can be regarded as a bravais lattice with a basis, however in this thesis we have exclusively worked with the primitive unit cell. This choice has the drawback that it makes visualization of the lattice harder than in the case of a cubic unit cell with a basis, but the advantage is that it simplifies the computation.

The primitive translation vectors for a fcc lattice are[5]

$$\begin{aligned} \vec{a} &= \frac{n\,a_o}{2}(\hat{\mathbf{x}}+\hat{\mathbf{y}}) \\ \vec{b} &= \frac{n\,a_o}{2}(\hat{\mathbf{y}}+\hat{\mathbf{z}}) \\ \vec{c} &= \frac{n\,a_o}{2}(\hat{\mathbf{z}}+\hat{\mathbf{x}}) \end{aligned} \tag{1.1}$$

The reciprocal lattice vectors are defined by

$$\begin{aligned} \vec{A} &= \frac{\vec{b}\times\vec{c}}{\vec{a}.\vec{b}\times\vec{c}} \\ \vec{B} &= \frac{\vec{c}\times\vec{a}}{\vec{a}.\vec{b}\times\vec{c}} \\ \vec{C} &= \frac{\vec{a}\times\vec{b}}{\vec{a}.\vec{b}\times\vec{c}} \end{aligned} \tag{1.2}$$

Using Equation 1.2 for a fcc lattice, we find that

$$\begin{aligned} \vec{A} &= \frac{2\pi}{n\,a_o}(\hat{\mathbf{x}}+\hat{\mathbf{y}}-\hat{\mathbf{z}}) \\ \vec{B} &= \frac{2\pi}{n\,a_o}(-\hat{\mathbf{x}}+\hat{\mathbf{y}}+\hat{\mathbf{z}}) \\ \vec{C} &= \frac{2\pi}{n\,a_o}(\hat{\mathbf{x}}-\hat{\mathbf{y}}+\hat{\mathbf{z}}) \end{aligned} \tag{1.3}$$

As can be seen from Equation 1.3, the reciprocal lattice of a fcc lattice is a bcc

[5] na_o is the lattice constant for the $n \times n \times n$ supercell

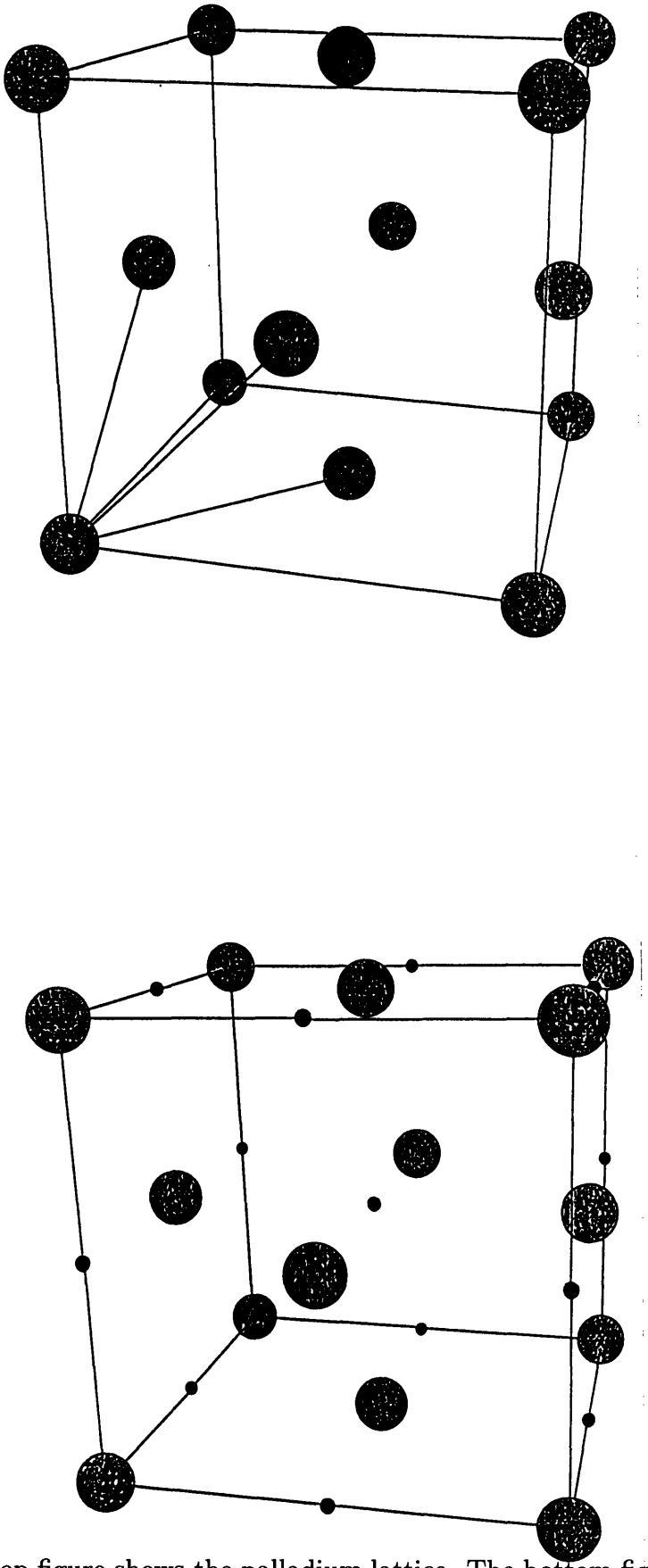

Figure 1-1: The top figure shows the palladium lattice. The bottom figure shows the palladium hydride lattice.

structure. Thus a general vector in the reciprocal lattice is described by

$$\begin{aligned} \vec{K} &= k_1\vec{A} + k_2\vec{B} + k_3\vec{C} \\ &= \frac{2\pi}{na_o}[K_x, K_y, K_z] \end{aligned} \tag{1.4}$$

where k_1, k_2 and k_3 are integers. The bcc structure along with the shape of the first Brillouin zone is shown in Figure 1-2.

The Brillouin zone can be analytically described by the following set of equations (using the notation of Equation 1.4)

$$\begin{gathered} \pm K_x \pm K_y \pm K_z \leq \tfrac{3}{2} \\ |K_x| \leq 1 \qquad |K_y| \leq 1 \qquad |K_z| \leq 1 \end{gathered} \tag{1.5}$$

These set of equations just describe the region bounded by the different planes of the first Brillouin zone which are at a distance of $\frac{\sqrt{3}\pi}{na_o}$ and $\frac{2\pi}{na_o}$ from the origin.

1.2 Palladium deutride/hydride

Palladium hydride is a non-stoichiometric compound and is usually written as PdD_x or PdH_x, where x is a real number between 0 and 1. The deuterium(hydrogen) atoms occupy octahedral sites as shown in Figure 1-1. However very small concentrations of deuterium/hydrogen can be found at the tetrahedral site[6] as well.

Adding hydrogen to the fcc unit cell does not change the reciprocal lattice at all, because the primitive translation vectors defined by Equation 1.1 do not change with the addition of hydrogen. The only difference now is that the bravais Pd lattice becomes a lattice with a basis: the two elements of the basis are $\vec{b}_1 = \vec{0}$ and $\vec{b}_2 = \hat{\mathbf{x}}a_o/2$. For the $n \times n \times n$ PdD/H unit cell, these results are generalized as follows:

1. The shape of the first Brillouin zone does not change, but the volume is scaled down by a factor of n^3 (each linear dimension is scaled down by n)

[6] a tetrahedral site is at $a_o(1/4, 1/4/1, 4)$

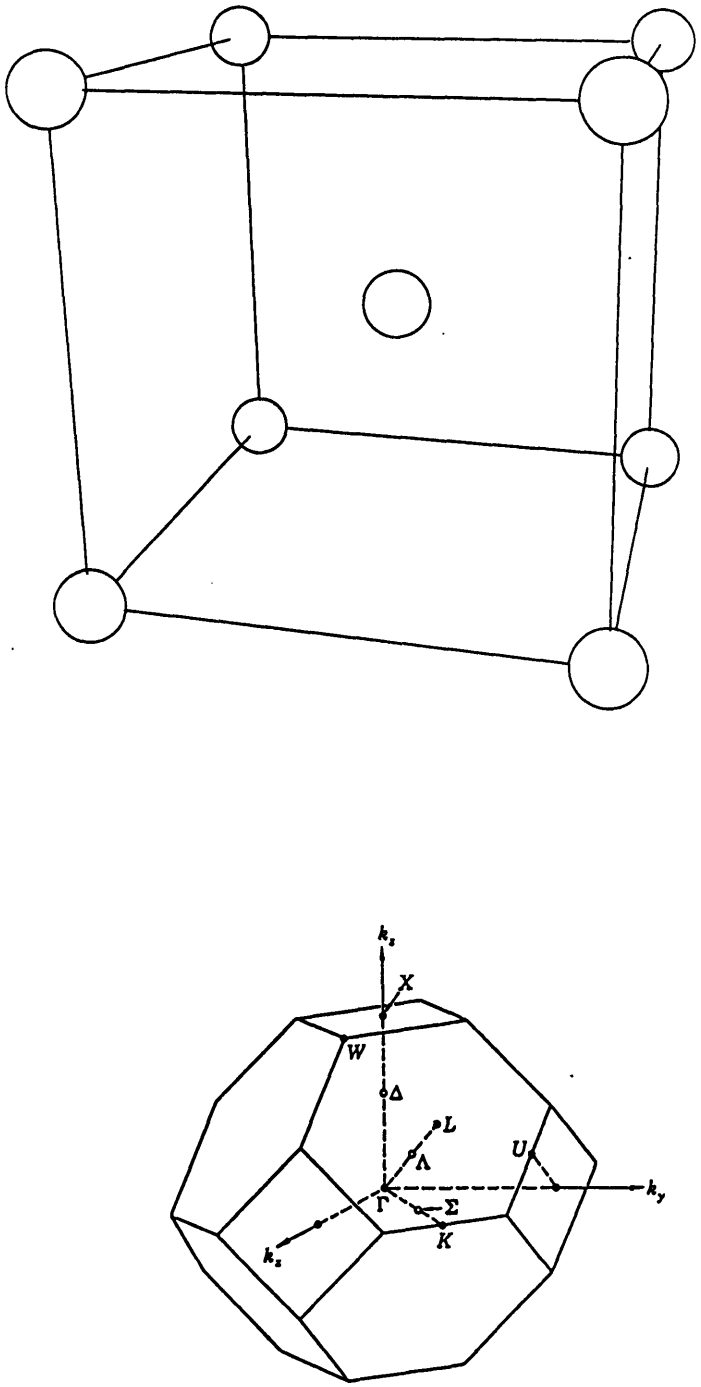

Figure 1-2: The top figure shows a bcc lattice. The bottom figure shows the first Brillouin zone of Pd or PdH

2. The bravais lattice becomes a lattice with a m dimensional basis, where the basis set consists of $\mathcal{B} = \vec{b}_1, \vec{b}_2, ...\vec{b}_m$. The $\vec{b}_i$ are the positions of Pd or D/H atoms with respect to a certain convenient point in the unit cell(usually taken to be the position vector of some Pd atom)

The first result is important in these calculations. The reason is that we can expand/contract our unit cell or add/subtract atoms to the unit cell, without any fear of changing the shape of our Brillouin zone. Thus any symmetries which are present in the Brillouin zone of PdD/H are also present in the Brillouin zone of our expanded and quasi-disordered supercell. The reason why we use the expression, "quasi-disordered", is that our lattice is really an ordered lattice (with translational symmetry and bloch wavefunctions) because we are using periodic boundary conditions to do all our calculations. However assuming that the interaction between the atoms is localized (to first or second nearest neighbors) the hope is that if we use a "large enough" unit cell with Pd vacancies, it will be a "good" approximation to the disordered system. This does not take into account the possibility of "clustering", and assumes that the randomness in a real PdD/H lattice is some perturbation about an average PdD/H ratio.

This concludes our introductory discussion of the method used to do these calculations and the Pd and PdD/H structure. In Chapter 2 we give the mathematical/physical background necessary to understand this thesis. All of the material in that chapter is very standard and can be found in any book on the theory of solids and lattice dynamics. In Chapter 3 we give the details of all the calculations which were done. And Chapter 4 contains conclusions and recommendations on further extensions of this thesis.

Chapter 2

Theoretical Background

The theory behind this computation is fairly simple and has been known for a long time [10]. It is just the basic application of Newton's laws to a lattice. In this chapter we establish the notational conventions, and outline the methods used for the calculation of the frequency spectrum.

2.1 The Dynamical Matrix

If the lattice has N unit cells, and s atoms per unit cell, then there are $3Ns$ equations of motion.

$$M_\alpha(l)\ddot{u}_\alpha(l) + \sum_{\beta l'} A_{\alpha\beta}(l,l')u_\beta(l') = 0 \tag{2.1}$$

In this equation $M_\alpha(l)$ is the mass of the atom with label α in cell l, $u_\alpha(l)$ is the displacement from the equilibrium position $\vec{R}(l)$, α and β label the $3s$ cartesian components of the s atoms in a unit cell, and $A_{\alpha\beta}(l,l')$ are the force constants. This equation can be solved by using normal coordinate waves defined by

$$d_j(\vec{k}) = N^{-1/2} \sum_{\alpha l} \sigma_\alpha^j(\vec{k}) \exp\left(i\vec{k}.\vec{R}_l^{\,\alpha}\right) M_\alpha^{1/2} u_\alpha(l) \tag{2.2}$$

There are N values of $\vec{k}$ in the first Brillouin zone and $3s$ branches specified by j. Using Equation 2.2 in Equation 2.1, the normalized dynamical matrix is calculated

to be

$$A_{\alpha\beta}(\vec{k}) = \frac{\sum_{l'} A_{\alpha\beta}(l,l') \exp i\vec{k}.(\vec{R}_l^{\alpha} - \vec{R}_{l'}^{\beta})}{(M_\alpha M_\beta)^{1/2}} \tag{2.3}$$

We are interested in finding the eigenvalues, $\omega_j(\vec{k})^2$, of this matrix. These are of primary importance because they can be used to calculate the density of states, $\rho(\omega)$. The knowledge of the density of states is required for most calculations in solid state physics.

The force constants, $A_{\alpha\beta}(l,l')$, are calculated by A. Rahman et al. [5] by fitting the phonon dispersion relations along symmetry directions. They are given in Table 2.1.

	1st Neighbor [110]	2nd Neighbor [200]
Pd-Pd	$\begin{pmatrix} 12104 & 18483 & 0 \\ 18483 & 12104 & 1184 \\ 0 & 0 & 1184 \end{pmatrix}$	$\begin{pmatrix} 4355 & 0 & 0 \\ 0 & -1484 & 0 \\ 0 & 0 & -1484 \end{pmatrix}$
H-H	$\begin{pmatrix} 269 & 1633 & 0 \\ 1633 & 269 & 0 \\ 0 & 0 & 1929 \end{pmatrix}$	
Pd-H	$\begin{pmatrix} 1697 & 0 & 0 \\ 0 & 2315 & 0 \\ 0 & 0 & 2315 \end{pmatrix}$	
D-D	$\begin{pmatrix} 269 & 1633 & 0 \\ 1633 & 269 & 0 \\ 0 & 0 & 1929 \end{pmatrix}$	
Pd-D	$\begin{pmatrix} 1414 & 0 & 0 \\ 0 & 1929 & 0 \\ 0 & 0 & 1929 \end{pmatrix}$	

Table 2.1: Force Constants in dyns/cm

The dispersion relations were experimentally found for a non-stoichiometric $PdH_{0.63}$ and $PdD_{0.63}$ by fitting the experimental data to a stoichiometric PdH and PdD respectively. There is a question as to the validity of such a scheme, since only the

phonon spectrum in the symmetry directions has been taken into account and the rest of the Brillouin zone has been ignored. Another problem which exists is that when we create a *Pd* vacancy then the lattice around the vacancy will relax. Thus the atoms around the vacancy will see a softer potential, and will be vibrating at a lower frequency as compared to the case of no *Pd* vacancy. No account was taken of these problems in these calculations. However the second problems can be solved using the Embedded Atom Method(EAM)[11, 12]. This point will be further considered in Chapter 4.

2.2 Building The Dynamical Matrix

As can be seen from Table 2.1, the force constants are given in terms of 3×3 matrices. The construction of the dynamical matrix (which is approximately a 150×150 matrix) from these 3×3 matrices requires some thought. We need to consider the following two points.

1. That the force constants are only given in $[1, 0, 0]$, $[2, 0, 0]$ and $[1, 1, 0]$ directions, whereas the atoms have nearest neighbors in other directions as well. So we need to figure out how the force constant matrices transform as we take into account neighbors in different directions.

2. We are using periodic boundary conditions for our computation. Nearest or second nearest neighbors of atoms in the unit cell may lie outside the unit cell. We need to find the atoms inside the supercell to which these atoms are equivalent to.

The transformations are fairly straightforward. And they are tabulated in Appendix A. The second problem can be solved by translating a neighbor ,β, of an atom, which lies outside the unit cell, by a lattice vector so that it is translated to some atom α inside the unit cell. The atom β is then equivalent to atom α. However this translation implies an additional phase factor of the form $exp(i\vec{k}.\vec{r}_{\alpha\beta})$ in the dynamical matrix where $\vec{r}_{\alpha\beta}$ is the distance between atoms α and β.

To help make these ideas clear, the dynamical matrix for a simple two dimensional example is explicitly constructed in Appendix B.

2.3 Speeding up the calculation

In all such supercell frequency spectrum calculations the speed of the algorithm used to diagonalize the dynamical matrix and compute the density of states is of the utmost importance. Simply dividing the Brillouin zone into a mesh and calculating the frequencies at all those points can simply be too slow a way to get the density of states. However our primary interest is in the phonon modes in the band gap. LAPACK has a routine which calculates the eigenvalues in a given region of the frequency spectrum. Surprisingly no real speed up was observed in comparison to the method in which the matrix was completely diagonalized. We will discuss the possibility of calculating eigenvalues in a given region of the spectrum in Chapter 4. However various other methods were implemented to speed up the calculation of the entire frequency spectrum. They are described in the subsections below.

2.3.1 Symmetries in the Brillouin zone

The first step we did to speed up the computation is to take into account all the symmetries present in the Brillouin zone. This was done by Kellermann [14]. The Brillouin zone has a 48 fold symmetry. That means that we only need to compute the eigenvalues in $1/48th$ of the Brillouin zone. All our calculations were done in the $1/48th$ of the Brillouin zone described by

$$K_z \leq K_y \leq K_x \qquad K_i \geq 0 \tag{2.4}$$

With this folding comes the problem of how to take into account the overcounting of frequencies for certain $\vec{k}$ values in the Brillouin zone. This problem and its solution in one dimension is shown in Figure 2-1

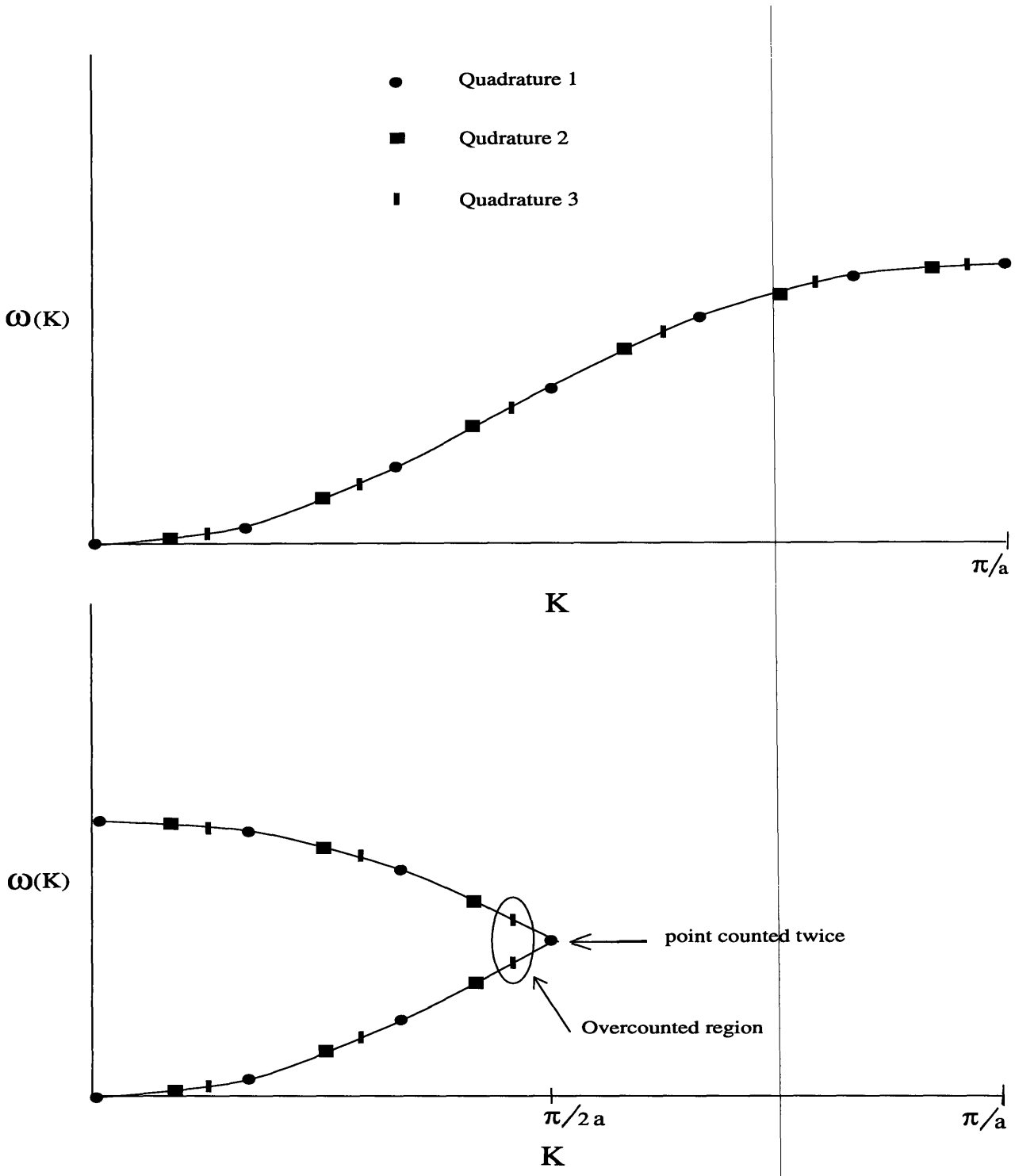

Figure 2-1: The top figure shows the dispersion relation for a cell with lattice constant a_o. The bottom figure shows the same dispersion relation for a supercell with lattice constant $2a_o$.

Quadrature 1[1] seems like the most natural way of dividing up the Brillouin zone. This works perfectly well for a $1 \times 1 \times 1$ cell. However if we change our lattice constant to $2a_o$, and we use the same quadrature, then we double count the frequency at $\pi/2a_o$. In one dimension this problem can be solved quite trivially by using Quadrature 2.

However the problem is not that trivial in higher dimensions. The reason can be seen by looking at Quadrature 3 in Figure 2-1. The only reason why Quadrature 2 works is that it is *exactly* half way between the points of Quadrature 1. However if we choose Quadrature 3, which is not half way between the points of Quadrature 1, then we are still overweighting a certain region of the frequency spectrum. From Figure 2-2 we see that if our Brillouin zone were of the form of a square (or a cube in 3 dimensions) then when we double our lattice constant we are still weighing all regions of the frequency spectrum equally. However in the non-square case, the distance between the boundary and the outer points of our quadrature varies. Hence when we double the lattice constant we will be weighing different regions of the frequency spectrum differently.

This problem is accentuated for us because we are dealing with a $n \times n \times n$ supercell. As explained in Chapter 2, the Brillouin zone folds on itself for the supercell structure e.g. for a $3 \times 3 \times 3$ cell the Brillouin zone reduces by a factor of 27. This causes a lot more inaccuracy as compared to the $1 \times 1 \times 1$ case.

There is a solution to this problem. The way we are doing our calculations is that we are assigning equal weights to all the points in the Brillouin zone.

$$\rho(\omega_i) = \sum_{\vec{K}(\omega=\omega_i)} \rho(\omega((\vec{K})))\Delta \tag{2.5}$$

However if we were to assign different weights to each of the points in our quadrature and use

$$\rho(\omega_i) = \sum_{\vec{K}(\omega=\omega_i)} \rho(\omega((\vec{K})))\Delta(\vec{K}) \tag{2.6}$$

then our problem is solved. However it is not obvious to us that how the weighting

[1] see Figure 2-1

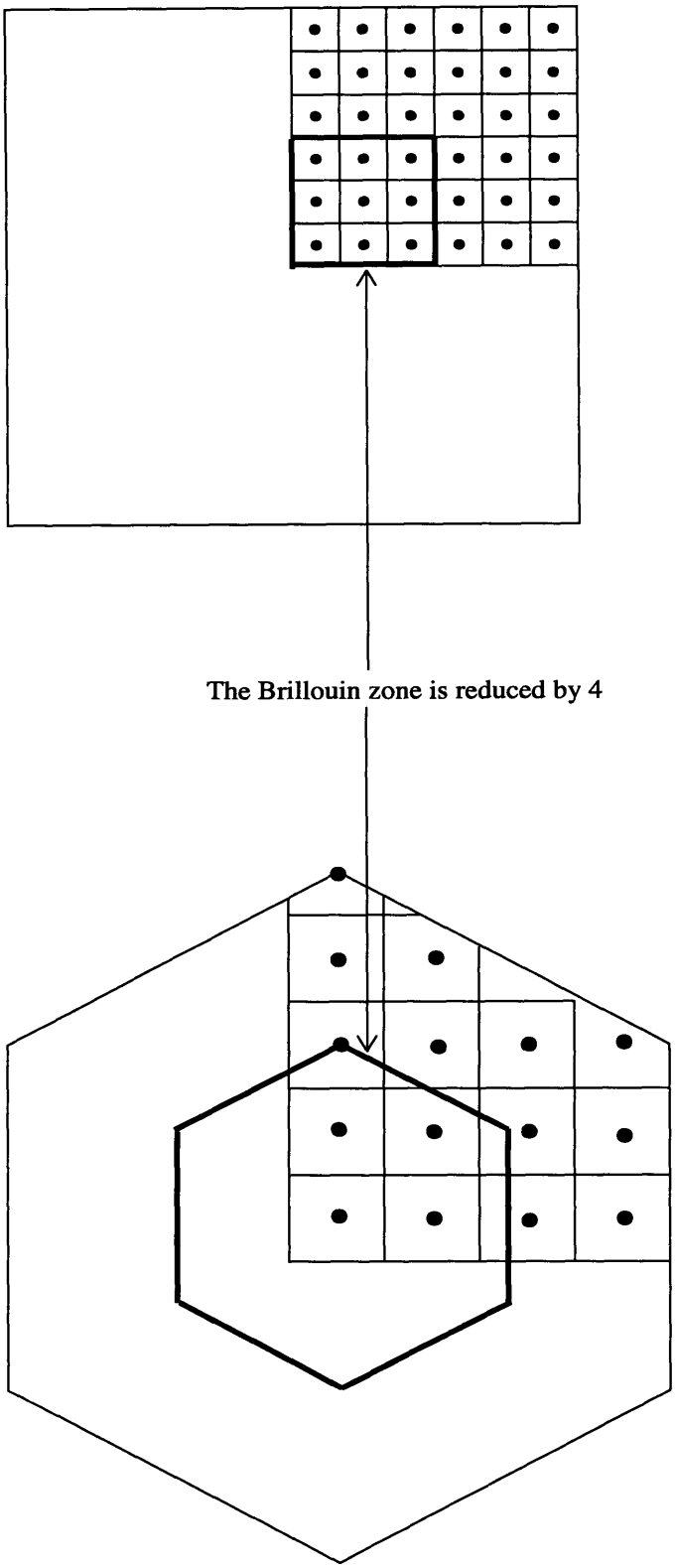

Figure 2-2: The figure shows two Brillouin zones. The square BZ does not cause any problems when scaling down. However the bottom one causes boundary problems.

scheme should be implemented. An easier solution exists, which is to ignore the problem! The reason why we can do that is because this overcounting is in some sense a surface term, and the interior points are like a volume term. If we diagonalize our matrix on a fine enough mesh to ensure that the surface term becomes negligible as compared to the volume term, then our overcounting error goes to zero. It takes about 500 points for the density of states to converge. Evaluating and diagonalizing this 150 $\times$ 150 matrix at approximately 500 points uses up a lot of computational time. Infact it took so long to diagonalize the matrix at those 500 points that some alternative ways had to be found. We tried two different ways of doing it.

- Non-degenerate first order perturbation theory
- Degenerate first order perturbation theory

Second order perturbation theory was not implemented, because by looking at various matrix elements it was found that mostly the second order terms are not important. However there are a few points where the second order terms are important. But at those points, perturbation theory is not valid because the second order corrections are more important than the first order corrections. Since the number of such points is so small, it was not worthwhile to implement second order perturbation theory.

The basic advantage of perturbation theory is that rather than diagonalizing the dynamical matrix on the entire mesh of points in the Brillouin zone, we diagonalize the matrix on a small subset of the mesh and then use perturbation theory to calculate the eigenvalues at the rest of the points. Another way to save time is to couple this perturbation calculation to linear extrapolation [16]. This reduces the amount of perturbation calculations. An outline of perturbation theory and linear extrapolation can be found in the next few subsections.

Perturbation theory is inherently an approximation. However, apart from the loss of accuracy, there is another disadvantage of using degenerate perturbation theory: we need to find the matrix, U, of eigenvectors to do perturbation theory. This means that we have to spend a lot more time on each diagonalization because computing

eigenvectors is very expensive. The differences in computational time between the two different diagonalization techniques is given in Table 1.1 in Chapter 1.

In the following subsections on perturbation theory, $D(\vec{q})$ is the dynamical matrix evaluated at the point $\vec{q}$ in the Brillouin zone. The dynamical matrix is diagonalized on a small number of points on a regularly spaced mesh ,$\mathcal{C}$, in the $1/48th$ Brillouin zone. Then perturbation theory or perturbation theory and linear extrapolation are used to find the eigenvalues on a fine mesh $\mathcal{F}$. (The next two sections essentially outline perturbation theory of matrix mechanics [13] with $D(\vec{q})$ as the unperturbed hamiltonian and $D(\vec{q}+\delta\vec{q})$ representing the perturbed hamiltonian).

2.3.2 Non-degenerate Perturbation Theory

This is the quickest and the crudest form of perturbation theory, but remarkably it gives excellent results[2] for the density of states [16]. U is the unitary matrix of eigenvectors of $D(\vec{q})$, where $\vec{q} \in \mathcal{C}$. Λ is the diagonal matrix of eigenvalues of $D(\vec{q})$. Then

$$U^{\dagger} D(\vec{q}) U = \Lambda \tag{2.7}$$

This is just the mathematical form of the statement that $D(\vec{q})$ is diagonal in the basis of its eigenvectors.[3]

Let $\vec{q}+\delta\vec{q} \in \mathcal{F}$. Then the perturbation matrix is given by

$$\Delta = D(\vec{q}+\delta\vec{q}) - D(\vec{q}) \tag{2.8}$$

From non-degenerate perturbation theory, the change in the eigenvalues ϵ_j is given by

$$\epsilon_j = \Delta'_{jj} + \sum_{k(\neq j)} \frac{(\Delta'_{jk})^2}{\Lambda_{jj} - \Lambda_{kk}} \tag{2.9}$$

[2] see Chapter 3

[3] ($U^{\dagger}DU$ just represents a change of basis)

where

$$\Delta' = U^{\dagger}\Delta U \tag{2.10}$$

As explained above, we did not implement second order perturbation theory. A major advantage of not implementing second order perturbation theory is that we do not need to calculate the off-diagonal elements of Δ'. This is a tremendous advantage since to calculate Δ' we need to do *three* very expensive matrix multiplications which is an $O(N^3)$ operation if we evaluate all the entries. However if we just calculate the diagonal elements the multiplication becomes an $O(N^2)$ operation.

2.3.3 Degenerate Perturbation Theory

Degenerate perturbation theory was used because as we increase the size of the supercell, the degeneracy in the eigenvalues increases. Exactly degenerate states are usually found in directions of high symmetry (like [110] or [111]). However as can be seen from Equation 2.9, if the eigenvalues are small on the scale defined by the square of the matrix element, then the second order correction is much larger than the first order correction: this signals the breakdown of perturbation theory. In that case degenerate perturbation theory has to be used [4].

Let us assume that eigenvalues $i_1, i_2 ..... i_r$ are found to be degenerate. Then to do degenerate perturbation theory we form the $r \times r$ submatrix, Γ, of Δ'. We then exactly diagonalize this submatrix using a unitary matrix V; the eigenvalues of this submatrix give us the first order perturbations in the eigenvalues of the degenerate eigenvalues.

To this most clearly we can do a simple 4×4 example assuming that eigenvalues

[4] see for example [13]

2 and 3 are degenerate.

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{pmatrix} \tag{2.11}$$

$$\Delta' = \begin{pmatrix} \Delta'_{11} & \Delta'_{12} & \Delta'_{13} & \Delta'_{14} \\ \Delta'_{21} & \Delta'_{22} & \Delta'_{23} & \Delta'_{24} \\ \Delta'_{31} & \Delta'_{32} & \Delta'_{33} & \Delta'_{34} \\ \Delta'_{41} & \Delta'_{42} & \Delta'_{43} & \Delta'_{44} \end{pmatrix} \tag{2.12}$$

$$\Gamma = \begin{pmatrix} \Delta'_{22} & \Delta'_{23} \\ \Delta'_{32} & \Delta'_{33} \end{pmatrix} \tag{2.13}$$

$$V\Gamma V^{\dagger} = \begin{pmatrix} \epsilon_1 & 0 \\ 0 & \epsilon_2 \end{pmatrix} \tag{2.14}$$

The ϵ_i is the perturbation in the second and third eigenvalues of Λ. Occasionally the degeneracy is still not lifted, and in that case we should expand our submatrix to include more terms, until the degeneracy is lifted. However this occurs so rarely that we can ignore it.

With the implementation of degenerate perturbation theory we lose some of the speed which we had gained using non-degenerate perturbation theory. Apart from having to form the matrix Γ (now for our calculation of Δ' we are no longer using $O(N^2)$ operations, but we are somewhere between $O(N^2)$ and $O(N^3)$ operations), we also have to exactly diagonalize it. The ultimate test of whether it is a useful way will depend on the speed of convergence.

2.3.4 Linear Extrapolation

Another way to save on computation time is to use linear extrapolation [16]. Let δq_i represent a change in the ith component of $\vec{q}$(i can range from 1 to 3). We define

$$\Delta = D(\vec{q} + \delta q_i) - D(\vec{q}) \tag{2.15}$$

and ω_j^o as the jth eigenvalue of $D(\vec{q})$ and ω_j^i as the jth eigenvalue of $D(\vec{q} + \delta q_i)$. With these definitions we can easily implement linear extrapolation.

$$\begin{aligned} \frac{\epsilon_j^i}{\delta q_i} &= \frac{(\omega_j^i)^2 - (\omega_j^o)^2}{\delta q_i} \\ &= \frac{\partial \omega_j^2}{\partial q_i} \\ &= 2\omega_j \frac{\partial \omega_j}{q_i} \end{aligned} \tag{2.16}$$

Now using Equation 2.16 we can use linear extrapolation to calculate

$$\omega_j(\vec{q} + \Delta\vec{q}) = \omega_j(\vec{q}) + \sum_{i=1}^{3} \frac{\partial \omega_j}{\partial q_i} \Delta q_i \tag{2.17}$$

This coupled with perturbation theory can be used to evaluate eigenvalues on the fine mesh $\mathcal{F}$. However the accuracy of the linear extrapolation is questionable. When we use perturbation theory coupled with linear extrapolation, we are introducing two sources of second order errors. The hope is that since they are both second order terms, non-degenerate perturbation theory with linear extrapolation will give reasonably accurate results in the shortest possible time.

All these methods were tried because simple diagonalization seems too slow for the $3 \times 3 \times 3$ or the $4 \times 4 \times 4$ cell. However, as we have discussed above, there are various competing factors and it is not clear at this stage whether any of the above mentioned methods will actually speed up the computation. The ultimate test for all these methods is that how fast the density of states calculation converges.

This issue of the speed of convergence, along with the results of density of state

calculations for different types of supercells, is discussed in the next chapter.

Chapter 3

Results

In this chapter we present the various results of our calculations done for a palladium deutride/hydride lattice with different types of vacancies. The code which was used to calculate these results is given in Appendix D.

3.1 Checks

Before doing the calculation for the imperfect lattice, we checked our code against simple known facts about the palladium hydride/deutride lattice: the dispersion relation along the symmetry directions and the density of states. These calculations were done for a $1 \times 1 \times 1$ case. Then inorder to further convince ourselves that our code was fault free, we expanded our unit cell from $1 \times 1 \times 1$ to $n \times n \times n$ where n ranged from $2 - 4$, and again calculated the density of states.

Due to the folding of the Brillouin zone, the phonon spectrum for the supercell would have looked very different from the $1 \times 1 \times 1$ cell. To get the original spectrum, we would have had to translate the $3N_s$(N_s is the number of atoms in the supercell) branches of the frequency spectrum by reciprocal lattice vectors. This would have been extremely tedious. Thus the phonon spectrum along symmetry directions was not calculated for the supercell.

A brief discussion of the results is given in the subsections below.

3.1.1 Phonon Spectrum

The phonon spectrum along symmetry directions of palladium deutride has been experimentally measured [7]. To check our code, we first calculated the dispersion relations along symmetry directions for a $1 \times 1 \times 1$ cell.[1]. The curves match very well, but there is nothing really surprising in this, since the force constants were calculated using the phonon-dispersion curves. It merely serves as a first check on our code.

3.1.2 Density of states

Next the phonon density of states was computed. As mentioned in Chapter 2, the first Brillouin zone has a 48 fold symmetry. As a first check on our calculations we calculated the phonon density of states in some of the 48 different Brillouin zones. All of these Brillouin zones gave the same answer for the density of states. Next we expanded our unit cell to a $n \times n \times n$ cell. We again recomputed the density of states. Since we can regard the PdH/D as being a $1 \times 1 \times 1$ cell or a $n \times n \times n$ cell[2], we would expect the density of states calculation to give the same result for a $n \times n \times n$ case as for a $1 \times 1 \times 1$ case. This was indeed found to be true. Thus this served as an additional check on our code.

Our hope was that when we expand the unit cell, we will need less iterations for the density of states calculation to converge. The reason for this is that since the Brillouin zone has become much smaller, when we diagonalize our dynamical matrix at each point of the reciprocal space of the supercell, we are infact sampling more of the Brillouin zone of the original cell. Although to some extent the criterion for convergence is arbitrary, it can be seen from Table 3.1 [3] that the number of iterations decreases at approximately the same rate as the growth of the cell size.

But we do not expect the rate to match exactly because the error of the boundary terms increases very rapidly as we increases the size of the unit cell.[4] This error tends

[1] $1 \times 1 \times 1$ is in terms of primitive unit cells and not under the assumptions of a cubic lattice with a basis

[2] n goes from 2 to 4

[3] All calculations were done without the use of perturbation theory or linear extrapolation

[4] see chapter 2 for a discussion

size of unit cell	iterations	time
$1 \times 1 \times 1$	80	250
$2 \times 2 \times 2$	40	3550
$3 \times 3 \times 3$	20	5700

Table 3.1: Comparison of convergence times and iterations for density of states calculation using the root sampling method

to slow down the rate of convergence as we increase the size of our cell.

Inspite of the decrease in iterations, the time for computation increases very rapidly, with cell size, and that has to do with the fact that diagonalization of a large matrix is an expensive operation. As can be seen from Table 3.2, the above results also hold true for perturbation theory coupled with linear extrapolation.

size of unit cell	iterations	time
$1 \times 1 \times 1$	20 20	650
$2 \times 2 \times 2$	10 20	951
$3 \times 3 \times 3$	10 10	1.2×10^4

Table 3.2: Comparison of convergence times and iterations for density of states calculation using non-degenerate perturbation theory with linear extrapolation

The results of our calculations for the density of states for different cells is given in Figure 3-1

Thus with these calculations we became reasonably sure that our code was correct. The next thing to do was to check the speed of the various methods used in our calculation of density of states.

3.2 Comparison Of Various Methods

We tried the various methods[5] to calculate the density of states. The reason why we implemented all these different methods was to save on computational time.

[5]these are explained in Chapter 2

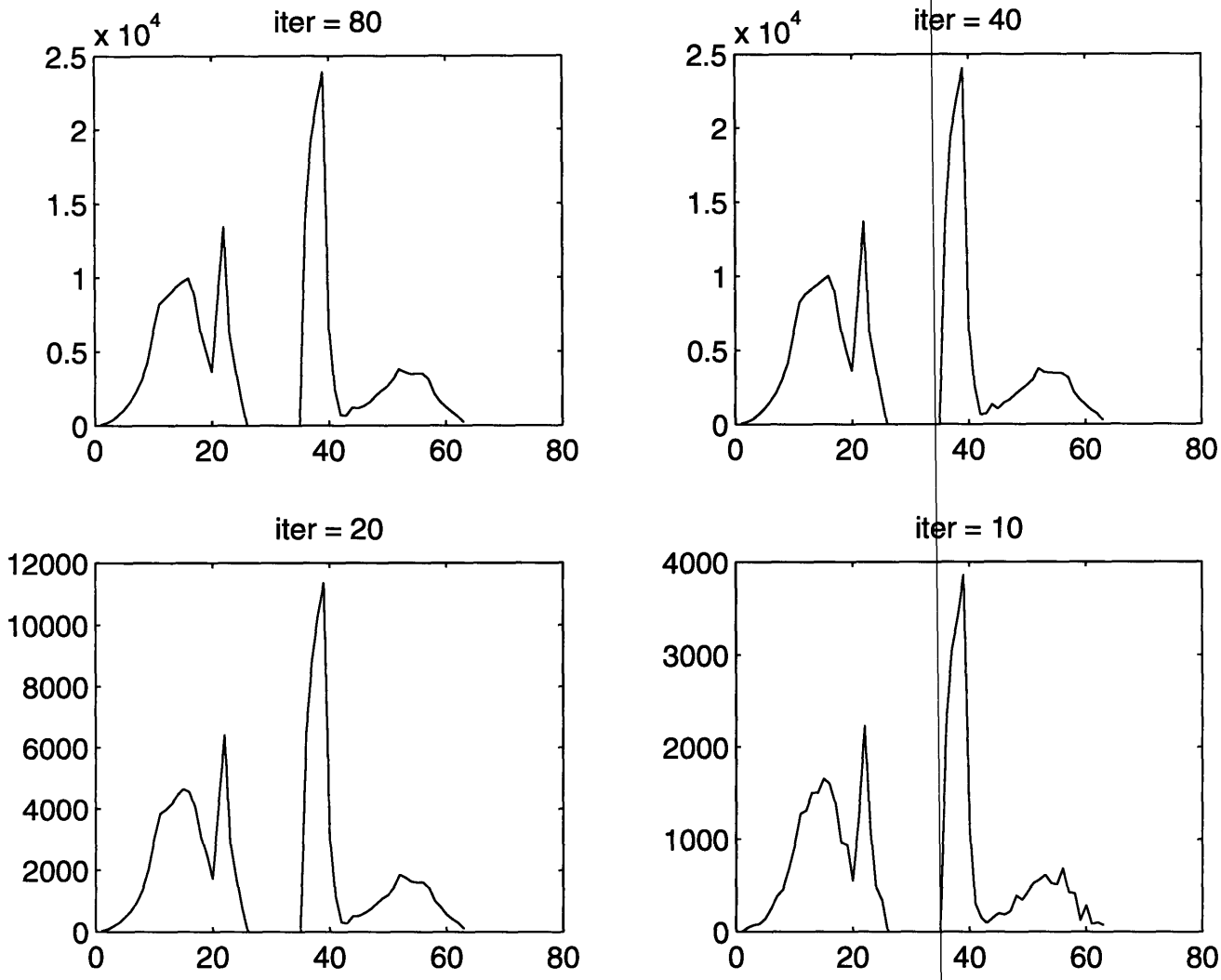

Figure 3-1: Density of state for different PdD supercells (in arbitrary units). Top left is $1 \times 1 \times 1$, top right is $2 \times 2 \times 2$, bottom left is $3 \times 3 \times 3$, bottom right is $4 \times 4 \times 4$

We can look at the results for the time taken for the various methods to converge for a $3 \times 3 \times 3$ cell. The reason why we are using a $3 \times 3 \times 3$ cell is because that is the cell size with which we will be doing most of our calculations.

NP	NDP	DP	NDPLE	DPLE
5.7×10^3	$> 7.2 \times 10^4$	7.2×10^4	1.2×10^4	$> 7.2 \times 10^4$

Table 3.3: comparison of convergence times for different methods

where

NP no perturbation theory or linear extrapolation was used

NDP non-degenerate perturbation theory used

DP degenerate perturbation theory used

NDPLE non-degenerate perturbation theory with linear extrapolation used

DPLE degenerate perturbation theory with linear extrapolation used

The details of these and other calculations can be found in Appendix C. However a few conclusions can be easily drawn from Table 3.3 [6]. They are:

- Perturbation theory without the use of linear extrapolation is useless. Perturbation theory alone is not fast enough to compensate for the errors it introduces into the calculation for the density of states. The real problem with perturbation theory is that it requires the computations of eigenvectors, which is an extremely expensive operation.

- The use of degenerate perturbation theory is quite useless in these calculations. Degenerate perturbation theory requires a lot more computation time as compared to the non-degenerate case, and it does not help the convergence rate to any great extent.

[6] see Appendix C for the meaning of NP,NDP,DP,NDPLE,DPLE

- Simple diagonalizations and non-degenerate perturbation theory with linear extrapolation are both reasonable ways of trying to go about solving the problem. It seems that the extra time required to find eigenvectors for perturbation theory more than compensates for the decrease in the number of points on which we have to diagonalize our matrix.

So in the end it seems that perturbative calculations are not as useful as we had hoped that they would be. Linear extrapolation along with non-degenerate perturbation theory is the only one, which competes with the simple diagonalization scheme. But even that is not as good as the simple diagonalization computation.

3.3 Dilute Limit

The next series of calculations were done for a pure palladium lattice with one deuterium/hydrogen atom at some octahedral site. This gave us the localized mode for deuterium/hydrogen. However it must be remembered that in this limit our force constants are not really valid. These force constants were obtained by fitting them to a a perfect palladium deutride/hydride lattice. In the dilute limit, when most of the deuteriums/hydrogens are absent, the lattice will considerably relax as compared to the perfect PdD/H case. When the lattice will relax, the palladium atoms will come closer to each other because there are no deuterium/hydrogen atoms to repel them. Thus we would expect the vibrational frequencies in a "real" dilute Pd lattice to be about 10% higher (a crude estimate based on the difference in the phonon spectrum of pure palladium along symmetry directions and the palladium spectrum calculated using these force constants).

As expected the localized mode for hydrogen vibrates at a higher frequency as compared to a deuterium atom.

The next calculation we did was to take out the palladium atom at the position $a_o(1,1,1)$, while keeping the deuterium or the hydrogen at $a_o(1,1,1/2)$. The deuterium has two different directions in which it can oscillate. It can oscillate in a plane perpendicular to the Pd vacancy (in this case the $x-y$ plane), or it can

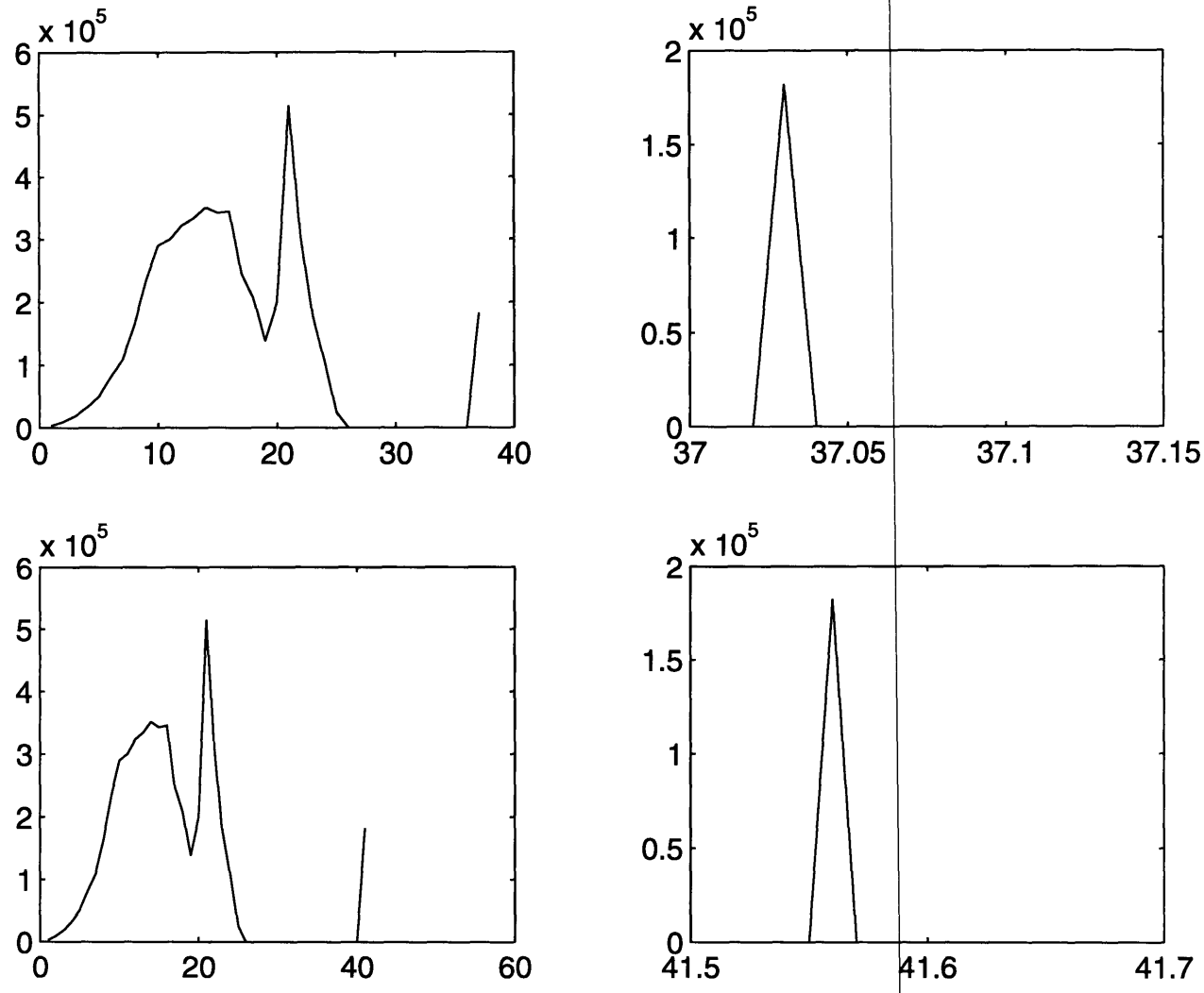

Figure 3-2: The top two figures show the density of states for a $3 \times 3 \times 3$ cell with a deuterium atom added at $a_o(1, 1, 1/2)$. The bottom figures show the density of states when hydrogen atom is added at the same position.

vibrate along the line defined by the Pd vacancy and the impurity atom. When it vibrates in the perpendicular plane, then it does not see a significantly softer potential as compared to the case of no Pd vacancy. However when it vibrates along the Pd vacancy-impurity line it sees a much softer potential, and hence the mode shifts significantly as compared to the perfect lattice.

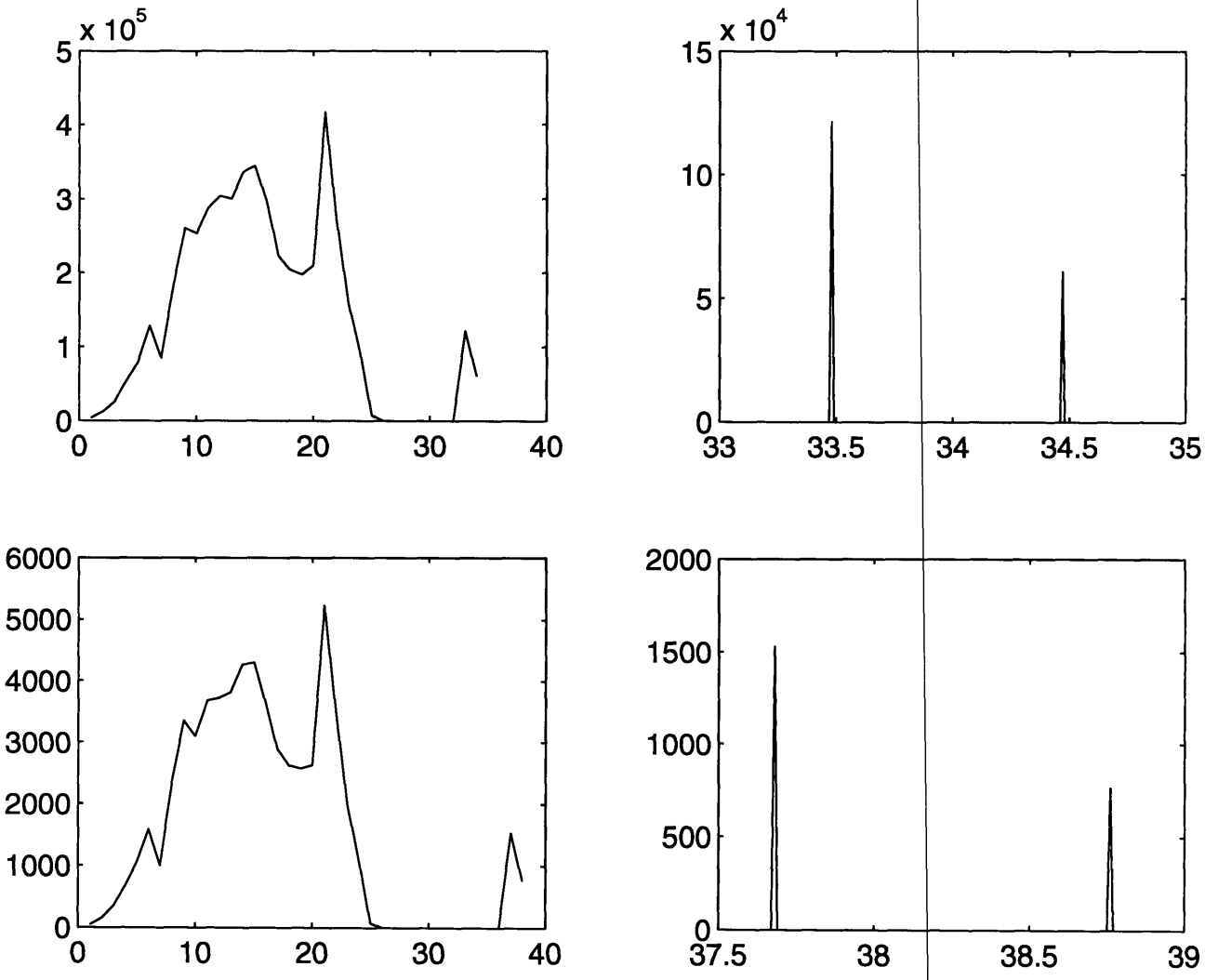

Figure 3-3: The top two figures show the density of states for a $3 \times 3 \times 3$ cell with a deuterium atom added at $a_o(1, 1, 1/2)$ and a Pd atom removed from $a_o(1, 1, 1)$. The bottom figures show the density of states for the same structure with hydrogen replacing the deuterium.

We also tried our calculation with Pd and H/D vacancies at different adjacent

sites. All of the calculations we did gave us exactly the same answers. Hence this served as an additional check on our code.

3.4 The Fukai Structure

Recently Fukai and Ōkuma have carried out experiments on Ni and Pd at high hydrogen pressure and temperatures of $\leq 1073K$. They found that there was an anamolous lattice contraction of the hydride, which they have attributed to the formation of Pd vacancies with a concentration of about $20 - 25\%$. We carried out the calculation of the phonon spectrum of the proposed palladium hydride/deutride structure in which every fourth Pd atom is missing from the perfect $NaCl$ like structure. The results of this calculation are shown in Figure 3-4.

3.5 Vacancy modes near a Pd vacancy

Our main interest in these calculations was to calculate the vacancy modes of palladium hydride/deutride. We created a Pd vacancy at $a_o(1,1,1)$ and observed the density of states. The results are shown in Figures 3-5 and 3-6.

The results are very interesting. The Pd vacancy has sufficiently softened up the potential for the impurity atoms that a large number of modes have shifted to the bottom of the optical band. We were hoping for the phonon modes to be in the band gap, however we have to remember that we have not taken lattice relaxation into account. Pd is a large atom. When we remove it from the cell, it will have a significant effect on the potential seen by the atoms around it. The hope is that when we take lattice relaxation into account, the phonon modes will end up in the band gap.

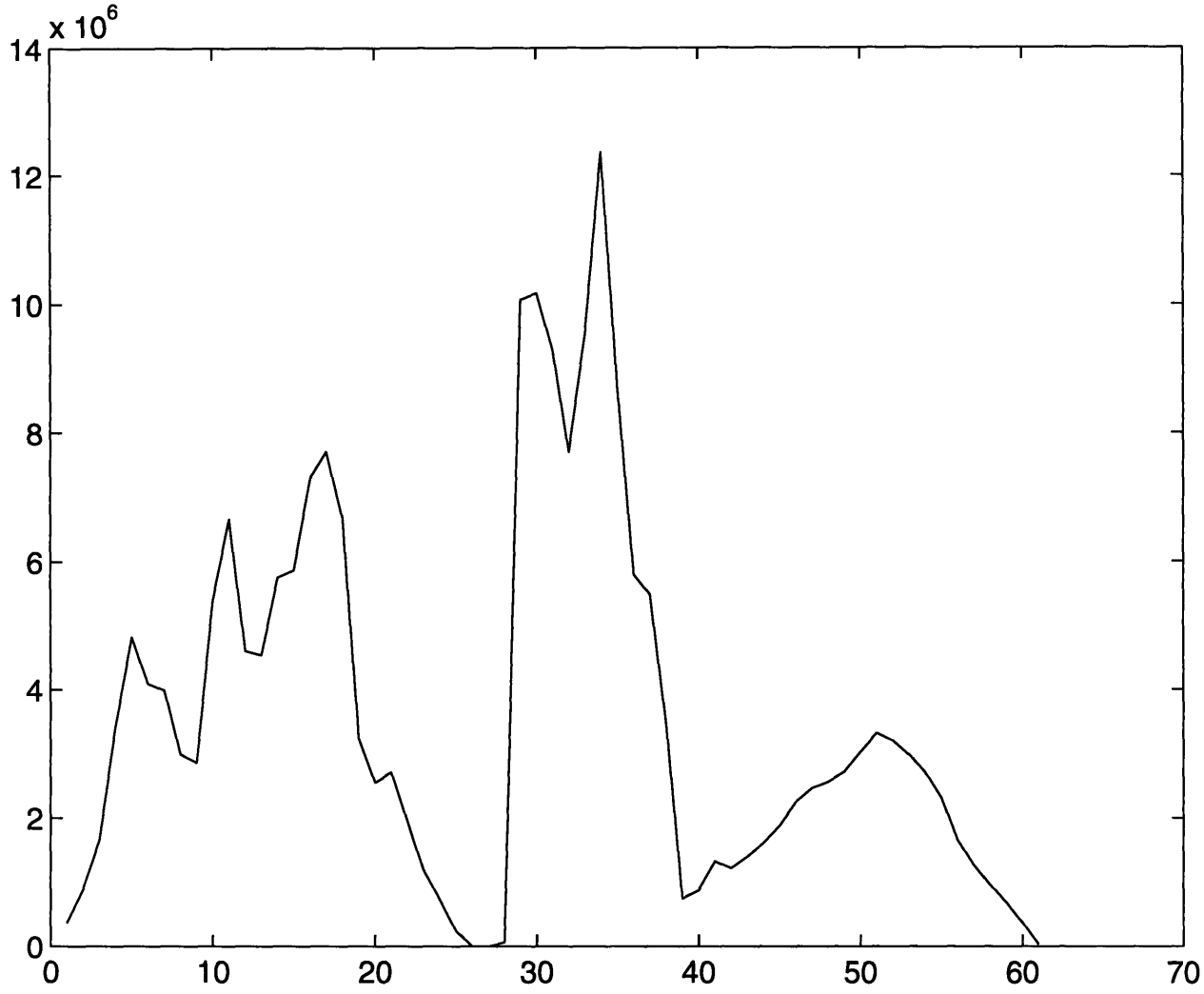

Figure 3-4: The figure shows the density of states for the Fukai structure for Palladium Deutride

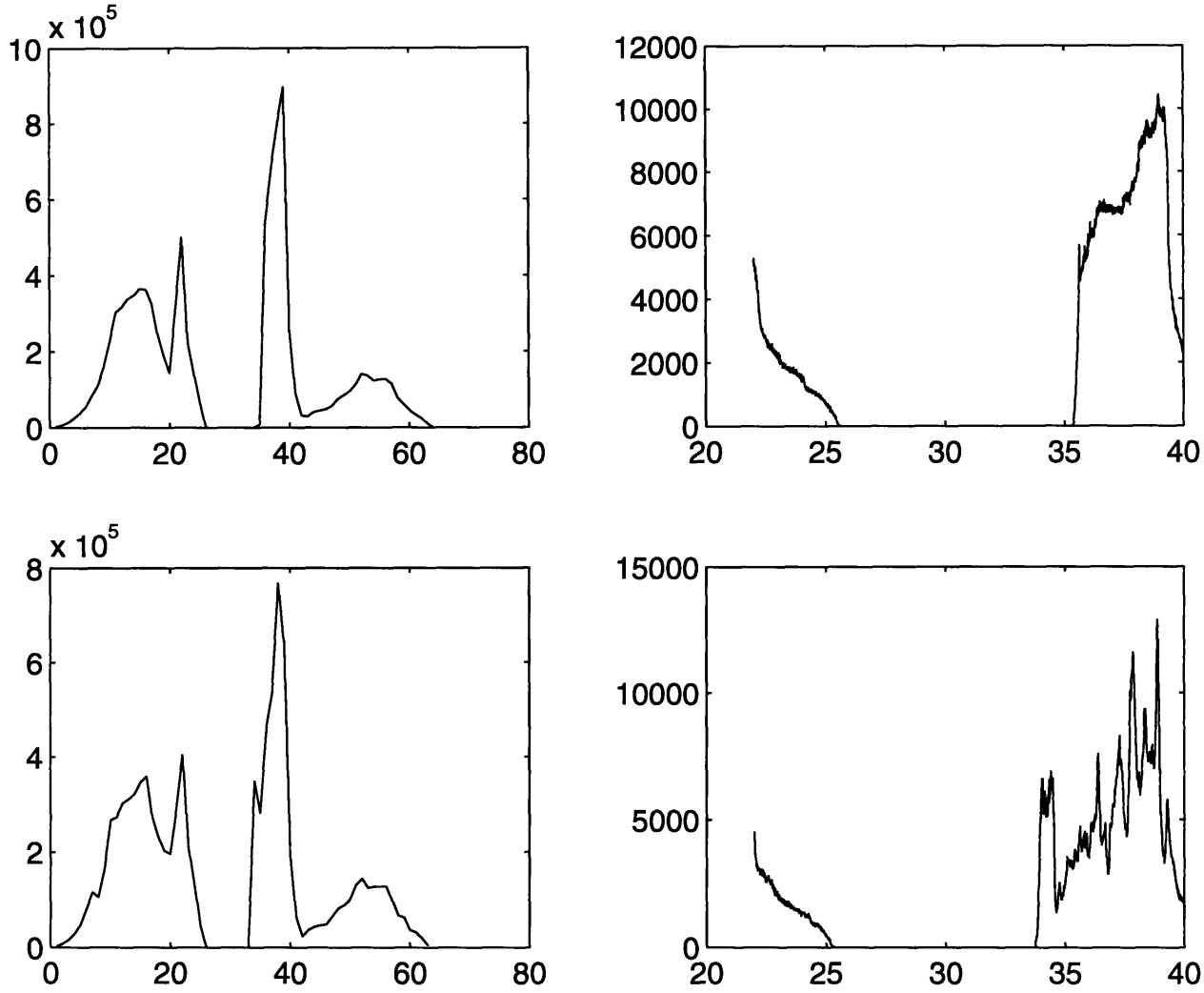

Figure 3-5: The top figures show the density of states for a perfect Palladium Deutride lattice. The bottom figure shows the density of states with Pd at $a_o(1,1,1)$ missing

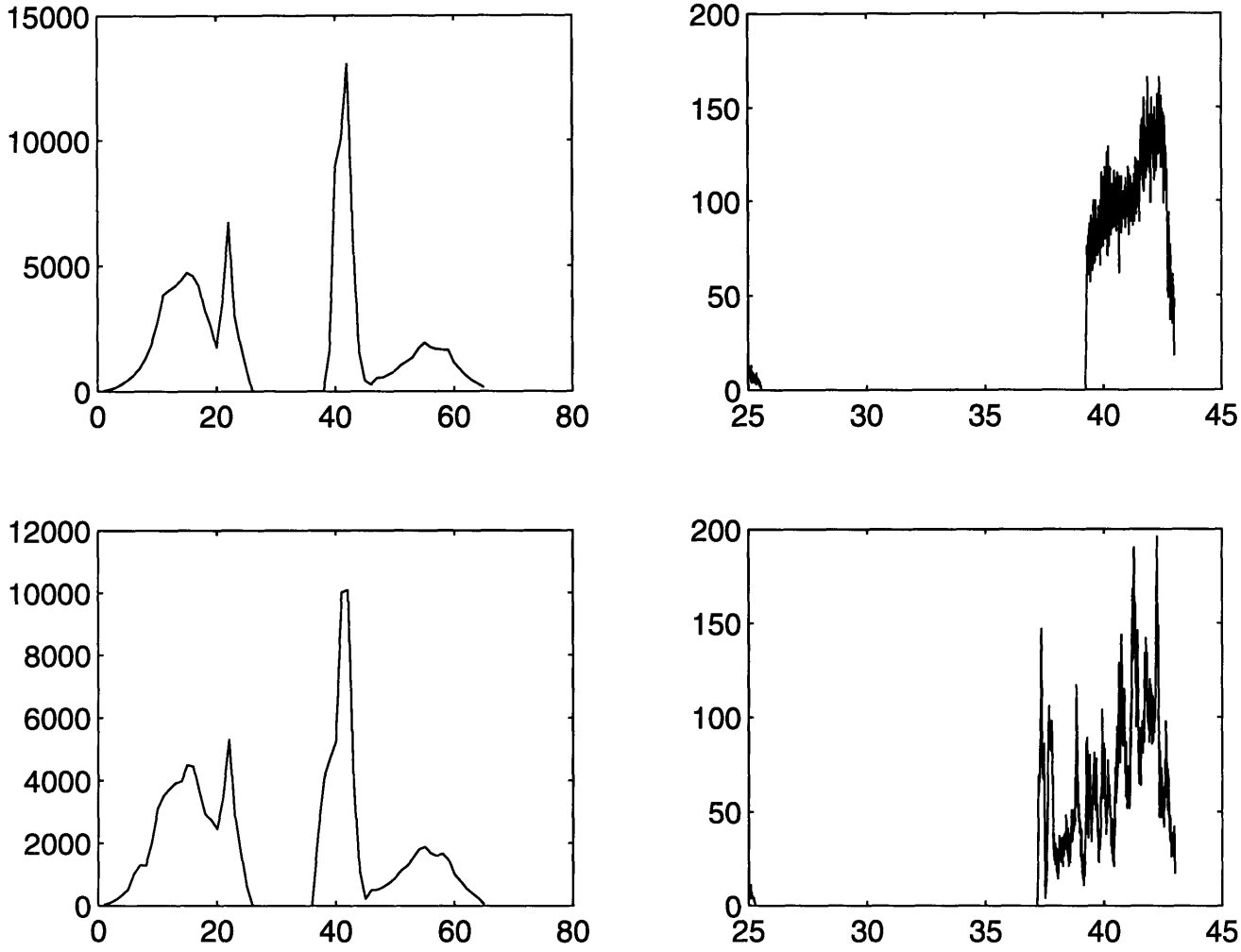

Figure 3-6: The top figures show the density of states for a perfect Palladium Hydride lattice. The bottom figure shows the density of states with Pd at $a_o(1,1,1)$ missing

Chapter 4

Conclusion and Extensions

In this thesis we wanted to compute the density of states of states for a palladium deutride/hydride supercell with arbitrary vacancies. The ideal which we wanted to achieve was to have a block box, in which we input the lattice structure at one end, and get the density of states at the output. This chapter summarizes the research, the conclusions we reached, and guidelines for future work. It also contains a section on the somewhat incomplete work done on lattice relaxation.

4.1 Main results

While calculating the eigenvalues of our normalized dynamical matrix we found that the simple diagonalization scheme was slow. So we also tried using simple perturbation techniques to compute the eigenvalues. The main results of these density of states calculations are:

1. Perturbation calculations are useless for the density of states calculations. Simple diagonalization schemes converge faster than any algorithm based on degenerate or non-degenerate perturbation theory.

2. There is an upper limit to the number of palladium vacancies which we can introduce into the structure.

3. When we introduce one palladium vacancy in a perfect palladium deutride/hydride structure, the density of states shows a pronounced peak towards the end of the optical band.

The first result differs from all the results which we have looked at [16]. This is probably due to the dramatic increase in computational power since the sixties and the seventies. It seems that simple perturbations techniques are no longer useful. Fancier perturbation methods might be useful for large supercells ($> 4 \times 4 \times 4$), because the root sampling method is only practically feasible for a relatively small supercells ($\leq 3 \times 3 \times 3$).

The second result very clearly shows the limitations of our calculations. Since we did not take lattice relaxation into account, we never expected our results to be accurate for the dilute limit. Just by comparing the experimentally determined phonon spectrum and the phonon spectrum calculated by using the force constants, we can roughly estimate that our frequencies were about $5 - 10\%$ lower than in an actual palladium lattice. However the interesting result that we found out was that we cannot remove palladium atoms from the lattice without taking lattice relaxation into account. We know that our calculations become unreliable when we introduce 25% palladium vacancies.[1] At this stage we do not know how inaccurate our results are. Further work need to be done.

Our main interest was in the phonon modes in the band gap. The third result is very promising. It shows that even without taking lattice relaxation into account, the vacancy phonon modes have almost fallen into the band gap. It seems very likely that the vacancy phonon modes will be in the band gap. We need to be somewhat cautious about this result, since in this calculation we are introducing Pd vacancies in the lattice. But it must be remembered that in this calculation we have only introduced a 3.5% vacancy. This is about 7 times smaller than the Fukai case. We would expect our force constant to be "reasonably" accurate in this case. Our belief is that even with lattice relaxation taken into account, the phonon modes will pile up

[1] See discussion in Chapter 3 of Fukai lattice

towards the end of the optical band or will fall into the band gap.

4.2 Relaxation

Two weeks prior to the thesis deadline we tried to incorporate lattice relaxation into the density of states calculation. Needless to say, the code could not be fully debugged by the time the thesis was due. This section contains a very brief discussion of the work done. As we have mentioned throughout this thesis, that an important source of error in the density of states calculation might be the relaxation effects. Relaxation in a lattice occurs when due to the addition or removal of an atom from the lattice at a site x, the host atoms around the site x, move to minimize the energy (or zero out the mean forces on each of the atoms). The way we propose to solve this problem is by using the Embedded Atom Method [11, 12]. In this section we summarize the theory and the results of our calculations with the Embedded Atom Method.

4.2.1 Embedded Atom Method

The Embedded Atom Method is an extremely simple way of taking into account the energy of a lattice. The theoretical motivation of the Embedded Atom Method comes from the density functional theory [17] and the effective medium approach [18]. The energy of the lattice is given by

$$\sum_i F_i(\rho_{h,i}) + \frac{1}{2}\sum_{i \neq j} \phi_{ij}(R_{ij}) \tag{4.1}$$

where F_i is an embedding function, $\rho_{h,i}$ is the host electron density at site i, and phi_{ij} is the doubly screened pair potential between atoms i and j, which are at a distance R_{ij} apart. The details can be found in the references cited above.

4.2.2 Minimization

Using the conjugate gradients algorithm we tried to minimize the energy of the lattice. The basic idea was that in the presence of a vacancy the atoms will position themselves

such that the total energy is minimized. So if we minimize the energy we will be able to find the equilibrium positions of the atoms. Then we can find the new force constants by calculating the curvature at those equilibrium positions.

4.2.3 Results

The application of the conjugate gradients algorithm to the embedded atom method did not give very good results (the conjugate gradients algorithm was implemented as in Numerical Recipes). The main problem was that not enough time was spent on this, and the code was not completely bug free by the thesis deadline. The way we did the calculation was that we took a $3 \times 3 \times 3$ cell. Then we took out the Pd atom at (1,1,1), and made the nearest neighbors of the Pd atom (both Pd and H) mobile, and fixed the rest of the atoms in the lattice. Then we tried to minimize the energy functional with respect to the positions of the mobile atoms. However unfortunately the code did not converge in this case. That is where we had to leave it, because of shortage of time.

4.3 Further Research

There are two main problems which still remain in this thesis which need to be solved. They are:

- The speed of the algorithm for diagonalization.
- The relaxation of the lattice.

Due to the time it takes LAPACK to diagonalize a matrix, we were constrained to work with supercell which were $\leq 3 \times 3 \times 3$. The reason why we want to work with the largest possible supercells is that the larger the supercell, the less is the effect of the boundary conditions. Also we would eventually like to include the effects of disordered vacancies in our computation. This would require us to work with large supercells. However as we increase the size of the supercell, our dynamical matrix

becomes a sparse matrix (since any atom can interact with its neighbors only). As mentioned before, our primary interest is in the band gap. Thus we do not need to find the eigenvalues in the entire range of the spectrum. This small frequency window and the sparsity of the matrix can be exploited to our advantage. Thus we need to use specialized algorithms used to solve eigenvalue problems of large sparse matrices in a given region of the frequency spectrum. Fortunately Lanczos algorithms precisely do that.

The second problem needs to be solved if we are to accurately calculate the frequency spectrum of a palladium deutride/hydride lattice with Pd vacancies. There are two ways which we propose to estimate the change in force constants with vacancies. The first one is a "back of the envelope" method and the other one is based on density functional theory. They are the vacancy relaxation method of Girfalco and Weizer [15] and the Embedded Atom Method of Murray, Daw and Baskes[11]. Both these methods allow us to calculate the total energy of the lattice.

The way we propose to find the force constants of a lattice with vacancies is to minimize the total energy function of the supercell. This gives us the equilibrium positions of the palladium and deuterium/hydrogen in the presence of the vacancy. Using the fact that the force constants tell us the forces acting on one atom when the other atom is moved in some direction, we can then easily calculate the force constants by calculating all the nine second derivatives of the energy function.

With these additions to the present calculations, we should be able to calculate the frequency spectrum of lattices with disordered vacancies.

Appendix A

Force Constants For Different Directions

The force constants are in general given for [100] and [110] directions for fcc lattices. The force constants in other directions are very simple transforms of the matrices in the [100] and [110] directions. The matrices for the nearest neighbor $Pd-Pd$ and the nearest neighbor $D-D$ or $H-H$ interaction have the same structure. Similarly the matrices for second nearest neighbor $Pd-Pd$ and nearest neighbor $Pd-D$ or $Pd-H$ have the same structure. Thus we are interested in transformations of matrices of two types only. They are

$$\begin{pmatrix} a & b & 0 \\ b & a & 0 \\ 0 & 0 & d \end{pmatrix}$$

$$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & b \end{pmatrix} \tag{A.1}$$

where a, b and d are the appropriate force constants depending on which two atoms we are considering. The entries in the force constant matrix,$A_{\alpha\beta}$ tell us the force per unit length in the direction α when the neighboring atom is moved in the direction β. From this definition of the force constant matrix it is simple to figure out the

transformed matrices.

direction	matrix	direction	matrix
$[1\,1\,0] \equiv [\bar{1}\ \bar{1}\,0]$	$\begin{pmatrix} a & b & 0 \\ b & a & 0 \\ 0 & 0 & d \end{pmatrix}$	$[1\ \bar{1}\,0] \equiv [\bar{1}\,1\,0]$	$\begin{pmatrix} a & -b & 0 \\ -b & a & 0 \\ 0 & 0 & d \end{pmatrix}$
$[1\,0\,1] \equiv [\bar{1}\,0\ \bar{1}]$	$\begin{pmatrix} a & 0 & b \\ 0 & d & 0 \\ b & 0 & a \end{pmatrix}$	$[1\,0\ \bar{1}] \equiv [\bar{1}\,0\,1]$	$\begin{pmatrix} a & 0 & -b \\ 0 & d & 0 \\ -b & 0 & a \end{pmatrix}$
$[0\,1\,1] \equiv [0\ \bar{1}\ \bar{1}]$	$\begin{pmatrix} d & 0 & 0 \\ 0 & a & b \\ 0 & b & a \end{pmatrix}$	$[0\,1\ \bar{1}] \equiv [0\ \bar{1}\,1]$	$\begin{pmatrix} d & 0 & 0 \\ 0 & a & -b \\ 0 & -b & a \end{pmatrix}$

direction	matrix	direction	matrix
$[1\,0\,0] \equiv [\bar{1}\,0\,0]$	$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & b \end{pmatrix}$	$[0\,1\,0] \equiv [0\ \bar{1}\,0]$	$\begin{pmatrix} b & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & b \end{pmatrix}$
$[0\,0\,1] \equiv [0\,0\ \bar{1}]$	$\begin{pmatrix} b & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & a \end{pmatrix}$		

Appendix B

Construction Of The Dynamical Matrix

In this appendix we explicitly construct the dynamical matrix for a very simple two dimensional lattice with diagonal force constants. The structure for which we propose to construct the dynamical matrix is shown in Figure B-1

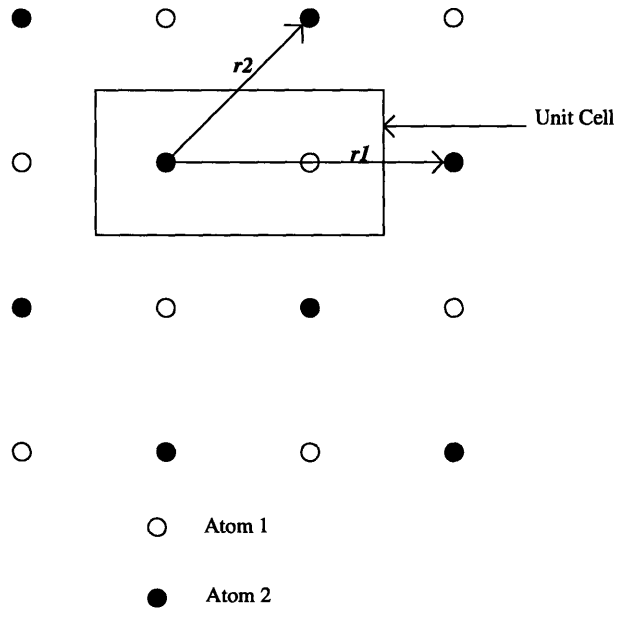

Figure B-1: A simple 2 dimensional structure.

The force constants for this structure are given (using the simplest model by ignoring the off-diagonal elements) in Table B.1

direction	matrix	direction	matrix
$[1,0] \equiv [-1,0]$	$\begin{pmatrix} a & 0 \\ 0 & d \end{pmatrix}$	$[0,1] \equiv [0,-1]$	$\begin{pmatrix} d & 0 \\ 0 & a \end{pmatrix}$

Table B.1: The toy force constants

Then the dynamical matrix is given by

$$\begin{pmatrix} \frac{2a}{m_1} & 0 & \frac{-2a}{\sqrt{m_1 m_2}} cos(\vec{K}.\vec{r}_1) & 0 \\ 0 & \frac{2d}{m_1} & 0 & \frac{-2d}{\sqrt{m_1 m_2}} cos(\vec{K}.\vec{r}_2) \\ \frac{-2a}{\sqrt{m_1 m_2}} cos(\vec{K}.\vec{r}_1) & 0 & \frac{2a}{m_2} & 0 \\ 0 & \frac{-2d}{\sqrt{m_1 m_2}} cos(\vec{K}.\vec{r}_2) & 0 & \frac{2d}{m_2} \end{pmatrix} \tag{B.1}$$

where the first row contains the forces acting on Atom 1 in the x direction (reading the first row from left to right) when

1. Atom 1 is moved in the x direction
2. Atom 1 is moved in the y direction
3. Atom 2 is moved in the x direction
4. Atom 2 is moved in the y direction

There are a few other points which should be noted

- In this case the dynamical matrix is symmetric. However for the general case, it is only hermitian.
- The *cos* terms come from adding two exponentials.

- The self terms (which are 2×2 matrices) are the sum of the corresponding row (or column) entries without the phase factors and are of the opposite sign as compared to the other entries.[1]

[1] by corresponding we mean that if we are considering the x direction forces, we add only the entries in the x rows(columns)

Appendix C

Detailed Results Of Calculations

In this Appendix we give the detailed results of all the calculations. When we used linear extrapolation, some of the frequencies turned out to be imaginary. Mistakes refers to the number of frequencies which were imaginary. Degeneracy refers to the number of degenerate eigenvalues. The definition of degeneracy is to some extent arbitrary. As explained in Chapter 2, it really depends on the ratio of the matrix element squared and the difference between the neighboring eigenvalues.

The next few pages show the results for the calculations of the density of states for the different number of iterations and various sizes of the supercell. In all of these graphs the x-axis is measured in units of meV and the density of states is in arbitrary units.

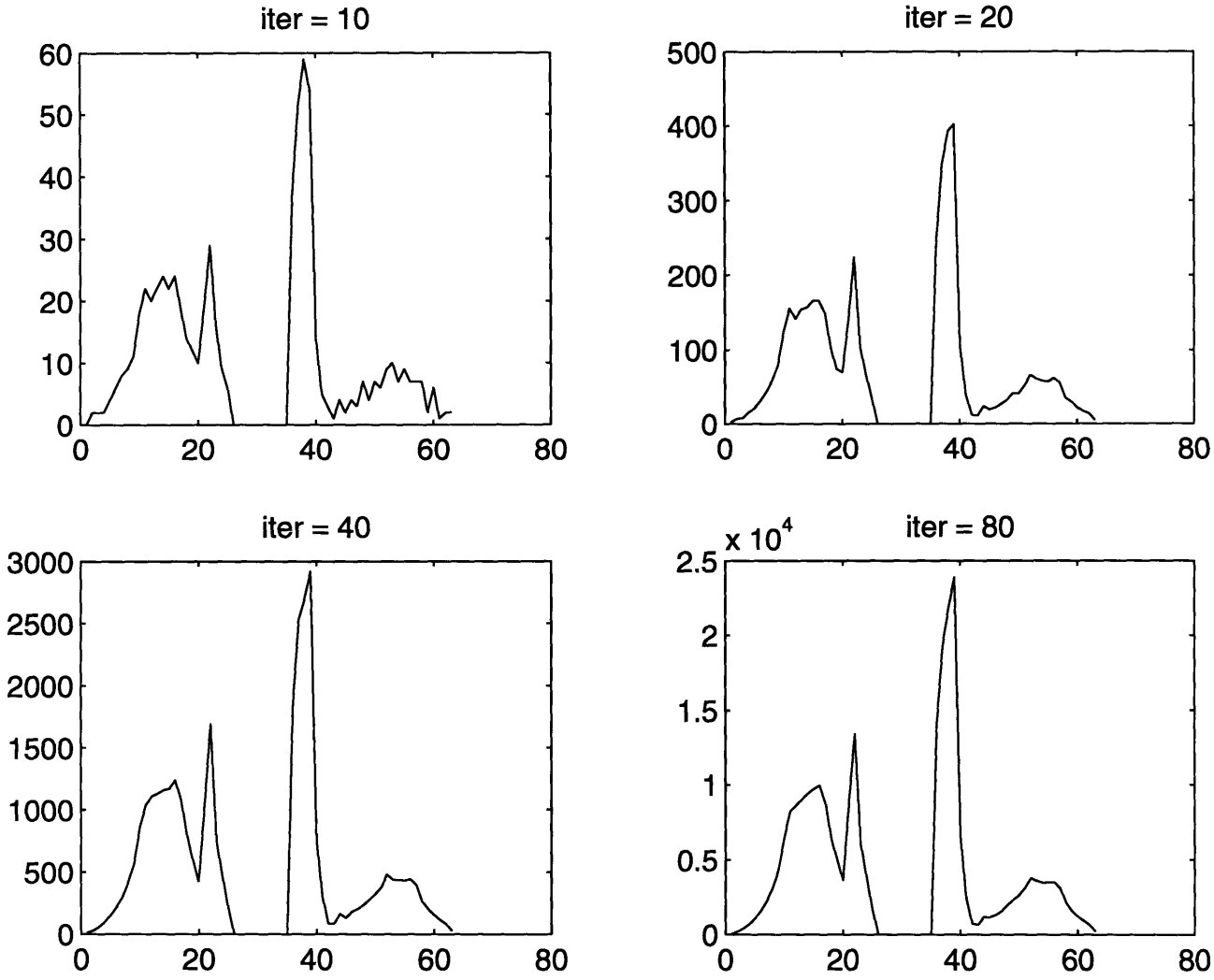

Figure C-1: Cell size is $1 \times 1 \times 1$. No perturbation theory or linear extrapolation is used.

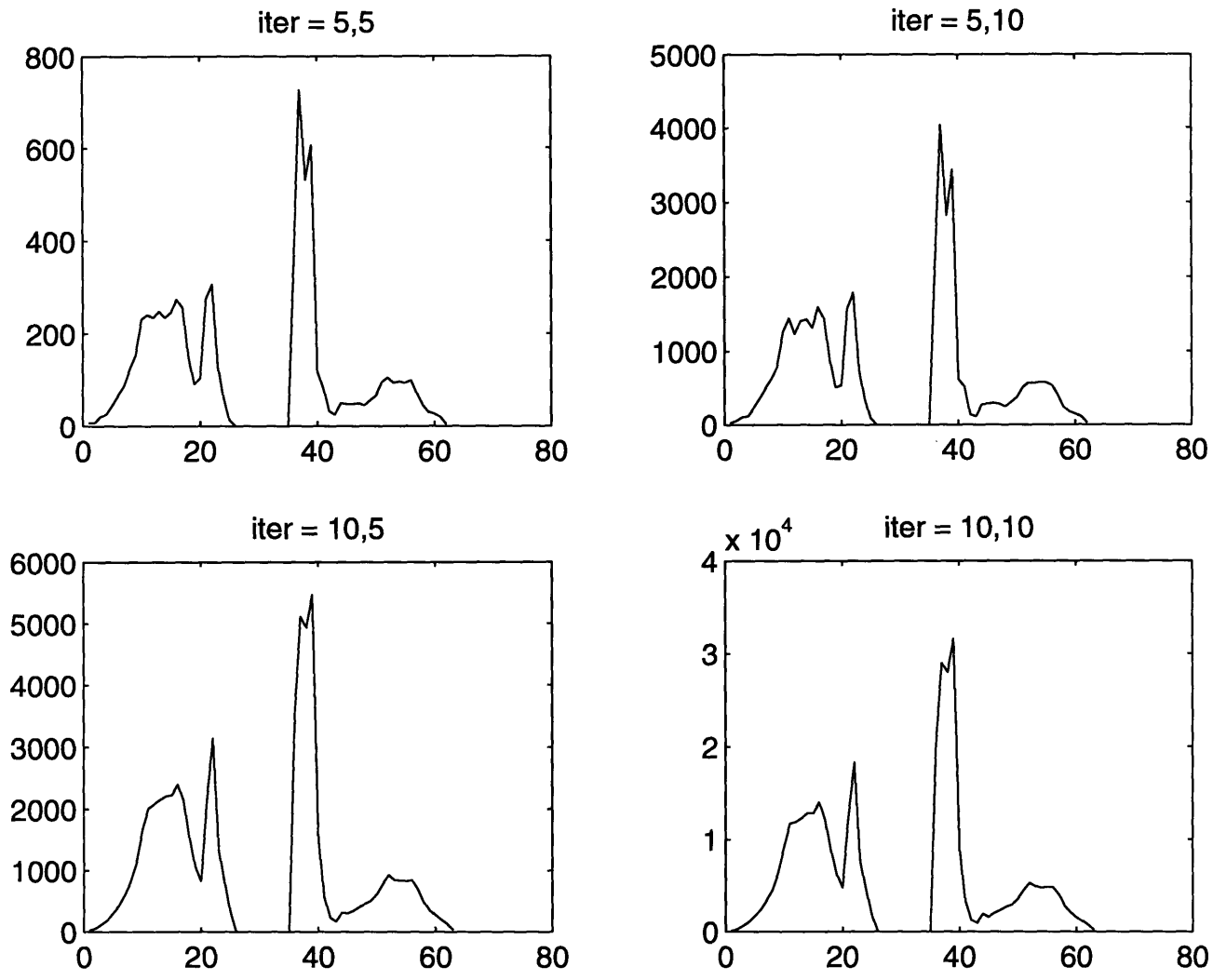

Figure C-2: Cell size is $1 \times 1 \times 1$. The graphs are calculated using non-degenerate perturbation theory

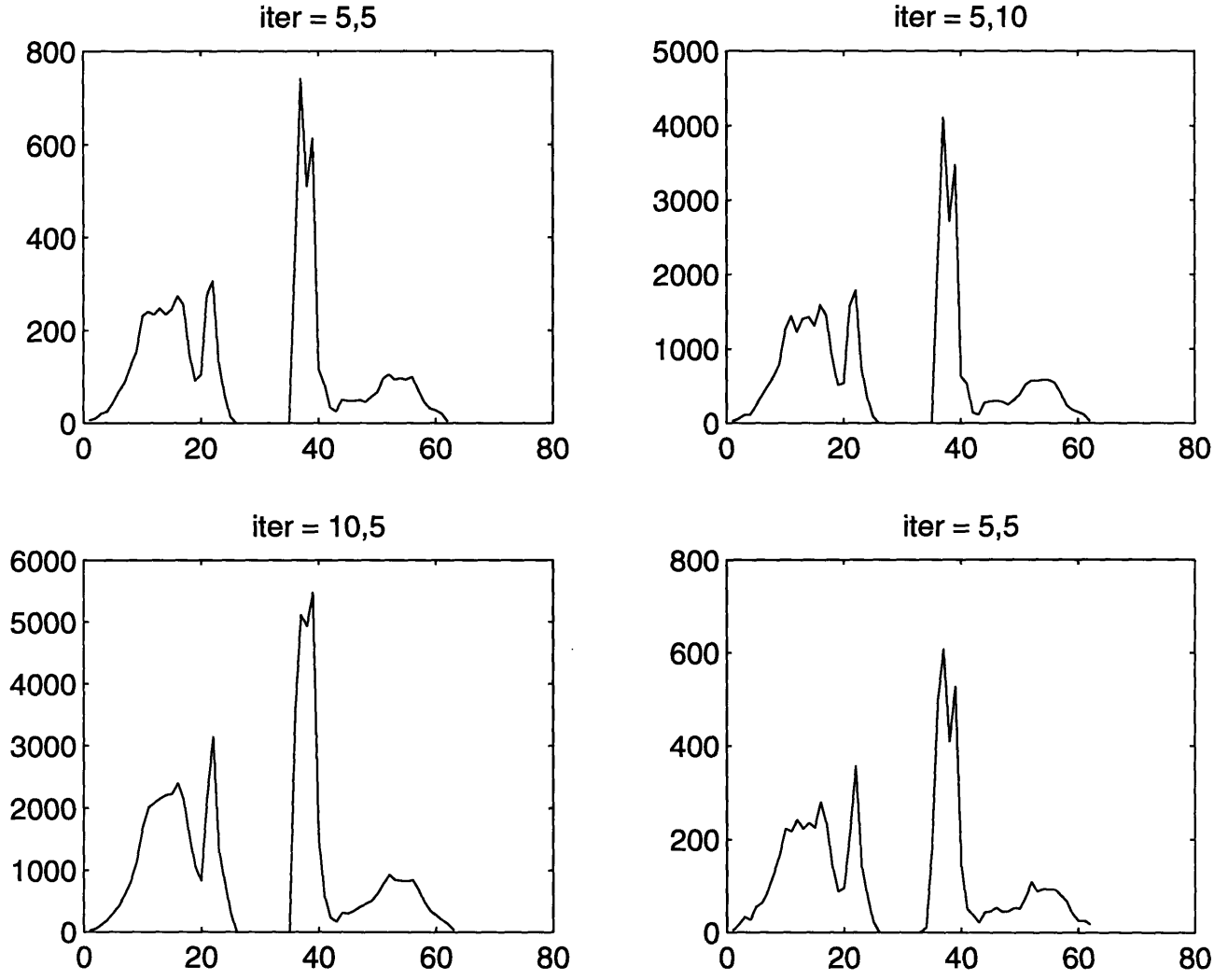

Figure C-3: Cell size is $1 \times 1 \times 1$. First 3 graphs are calculated using degenerate perturbation theory. The last graph is calculated using non-degenerate perturbation theory with linear extrapolation

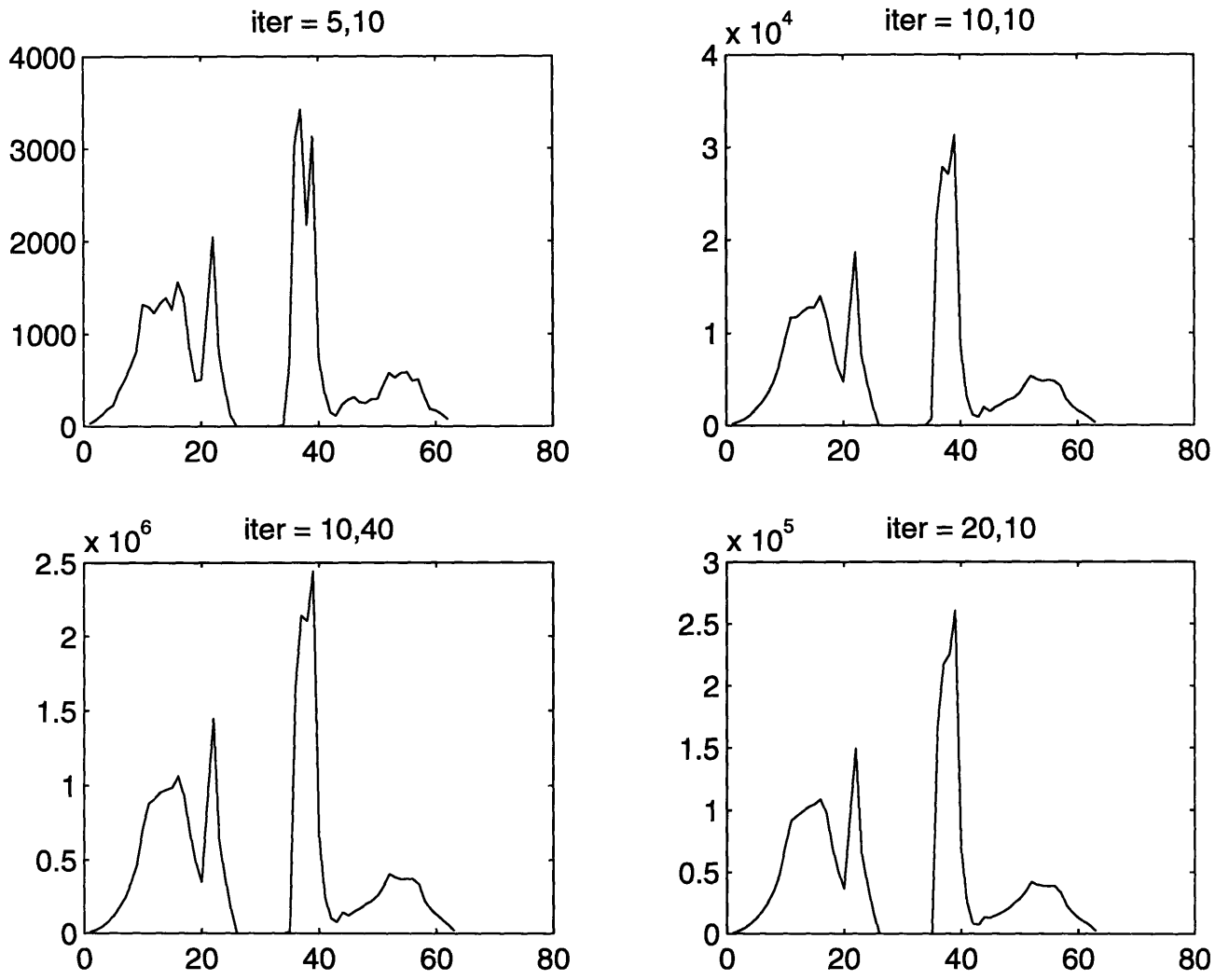

Figure C-4: Cell size is $1 \times 1 \times 1$. The graphs are calculated using non-degenerate perturbation theory with linear extrapolation

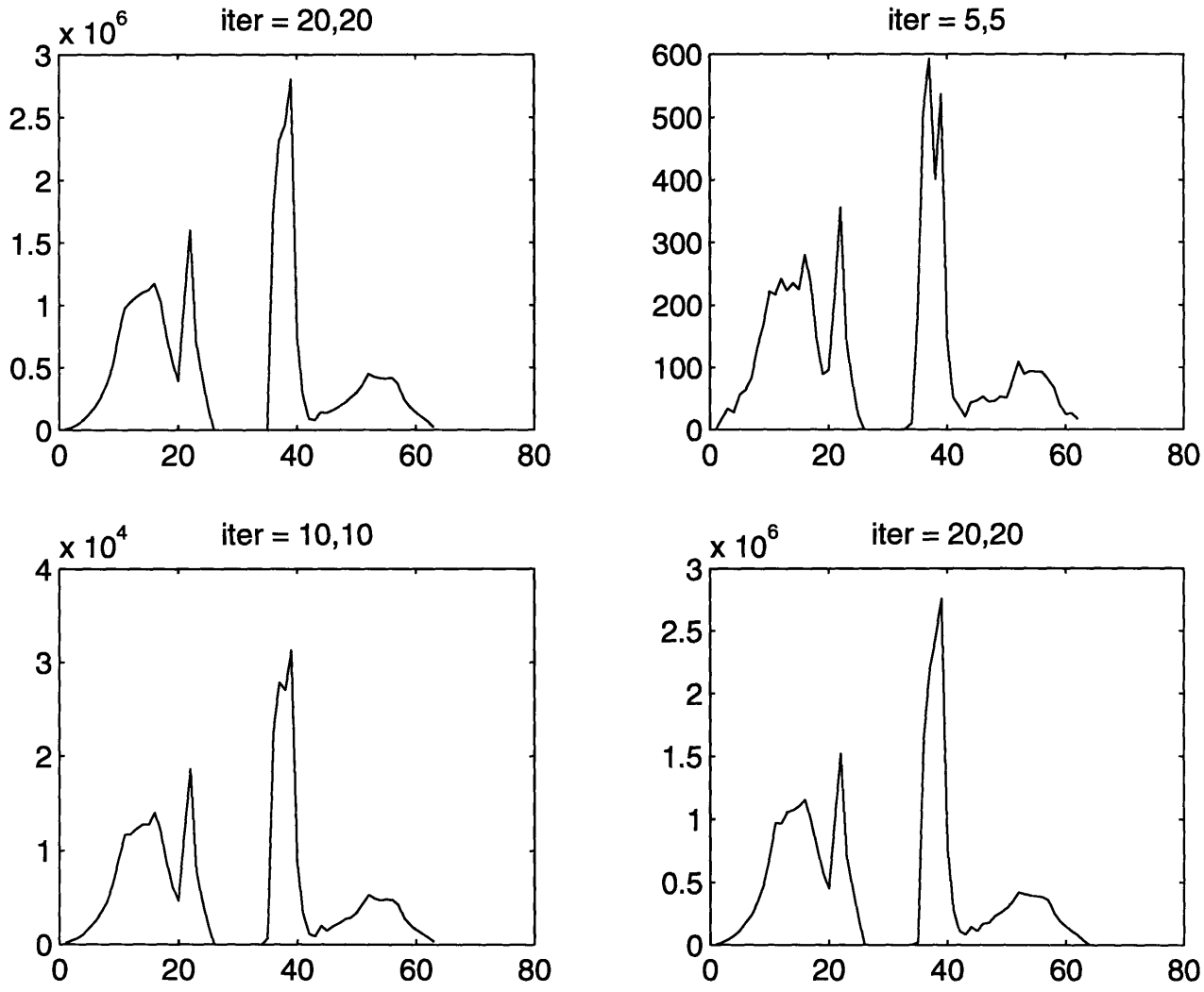

Figure C-5: Cell size is $1 \times 1 \times 1$. The first graph is calculated using non-degenerate perturbation theory with linear extrapolation. The last three graphs use degenerate perturbation theory with linear extrapolation

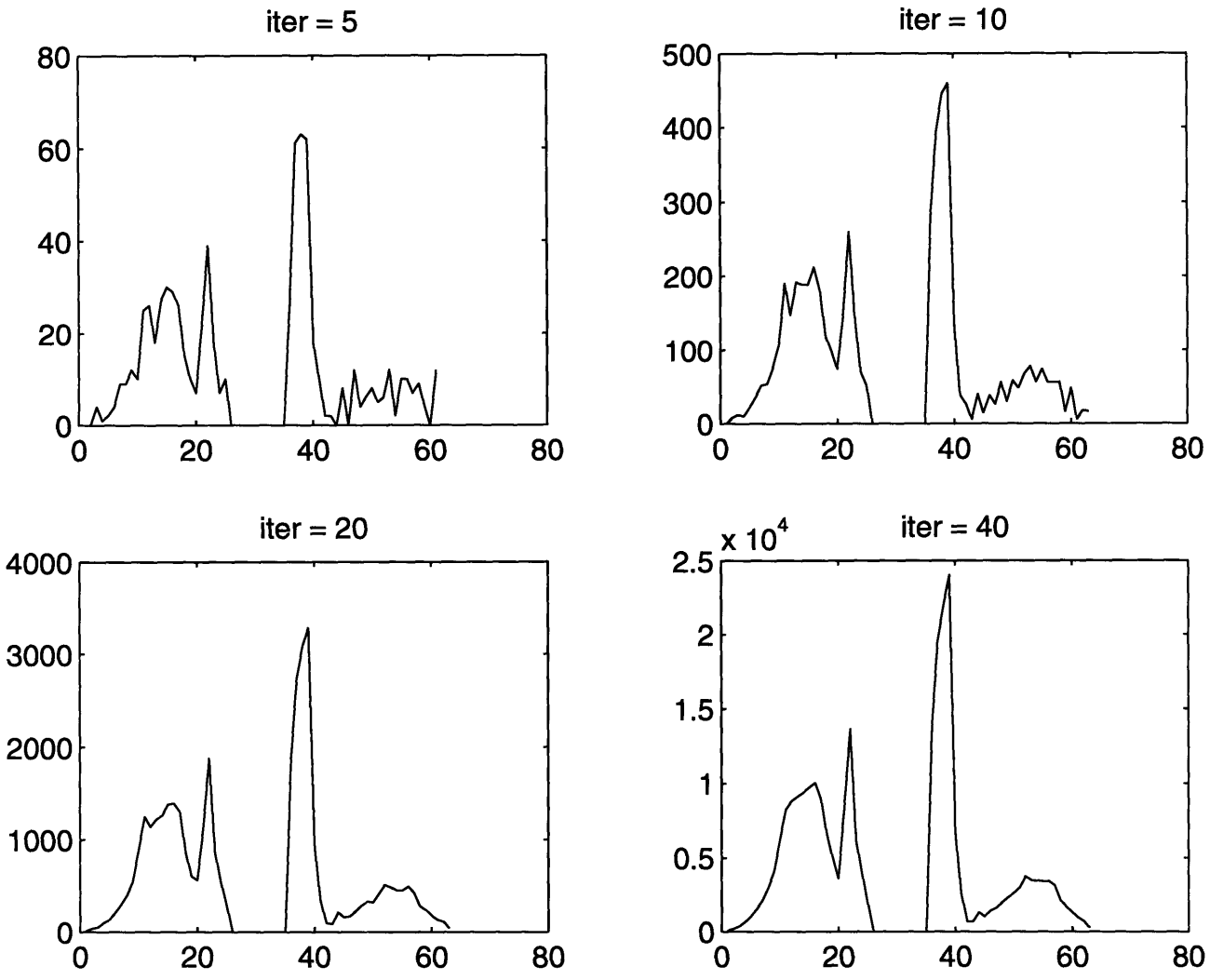

Figure C-6: Cell size is $2 \times 2 \times 2$. Perturbation theory or linear extrapolation were not used for the calculation of these graphs

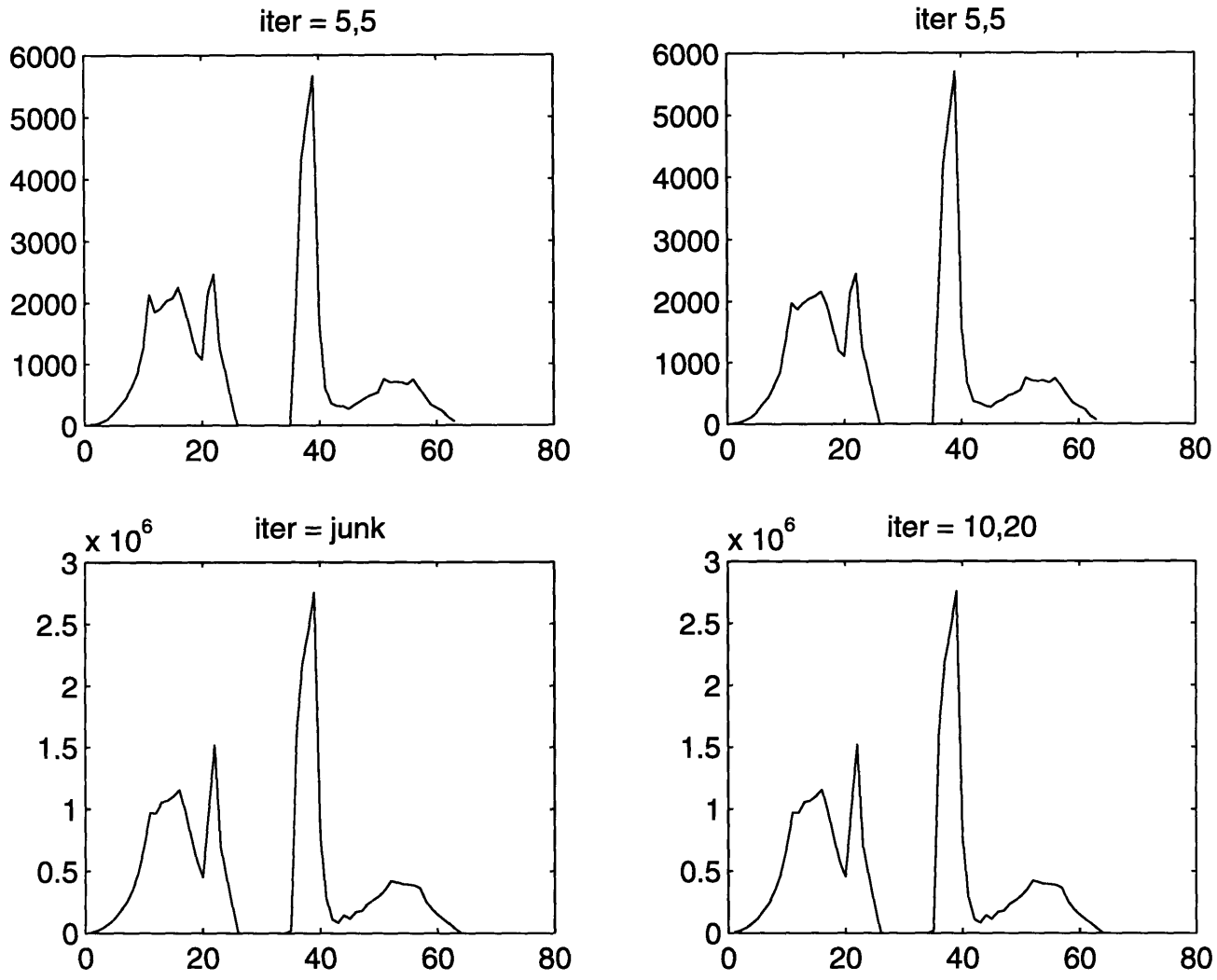

Figure C-7: Cell size is $2 \times 2 \times 2$. The first graph is calculated using non-degenerate perturbation theory, the second with degenerate perturbation theory, and the last two with non-degenerate perturbation theory with linear extrapolation

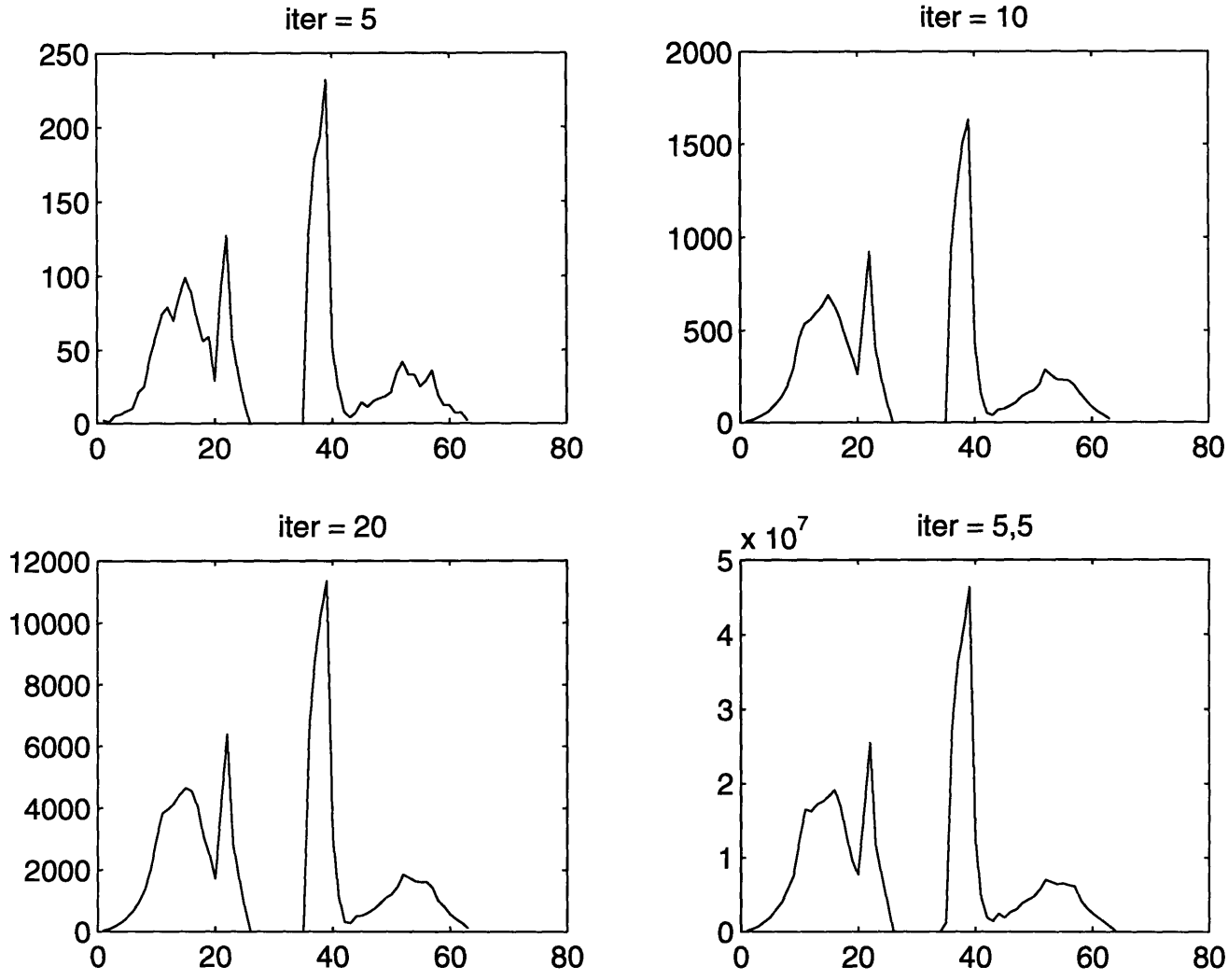

Figure C-8: Cell size is $3 \times 3 \times 3$. No Perturbation theory or linear extrapolation is used in the calculation of the first three graphs. The last graph uses non-degenerate perturbation theory with linear extrapolation

Method	Iterations	Time	$1 \times 1 \times 1$	$2 \times 2 \times 2$	$3 \times 3 \times 3$
NP	5	1.5	10.0	130	
	10	5.5	63.0	920	
	20	34.5	405	5.7×10^3	
	40	220	3600	$> 2.2 \times 10^5$	
	80	1700	-	-	
NDP	5,5	47	1200	> 3600	
	5,10	275	-	-	
	10,5	430	-	-	
	10,10	2580	-	-	
DP	5,5	53	2460	> 3600	
	5,10	340	-	-	
	10,5	535	-	-	
NDPLE	5,5	3.7	61	1730	
	5,10	4.3	70	-	
	5,20	11.4	120	-	
	10,5	19.8	441	-	
	10,10	24.5	1.2×10^4	-	
	10,20	86	951	-	
	10,40	520	-	-	
	20,10	167	-	-	
	20,20	650	-	-	
DPLE	5,5	4	113	3.7×10^6	
	10,10	27	-	-	

Table C.1: Comparison of various methods

Method	Iterations	Total Frequencies	Degeneracy	Mistakes
DP	5,5	7.0×10^3	4	0
	5,10	4.0×10^4	4	0
	10,5	6.0×10^3	12	0
NDPLE	5,5	7.3×10^3	-	33
	5,10	4.1×10^4	-	128
	5,20	4.2×10^5	-	603
	10,5	6.4×10^4	-	35
	10,10	3.6×10^5	-	128
	10,20	3.8×10^6	-	616
	10,40	2.8×10^7	-	3.4×10^3
	20,10	2.9×10^6	-	129
	20,20	3.1×10^7	-	615
DPLE	5,5	7.3×10^3	4	33
	10,10	3.6×10^5	12	112
	20,20	3.1×10^8	41	599

Table C.2: Mistakes for a $1 \times 1 \times 1$ cell

Method	Iterations	Total Frequencies	Degeneracy	Mistakes
DP	5,5	6.0×10^4	220	0
NDPLE	5,5	5.8×10^4	-	78
	5,10	3.3×10^5	-	285
	5,20	3.3×10^6	-	1.9×10^3
	10,5	5.1×10^5	-	1×10^3
	10,10			
	10,20	3.0×10^7	-	2.5×10^3
DPLE	5,5	5.8×10^4	221	82
	10,20	3×10^7	1.2×10^3	2.6×10^3

Table C.3: Mistakes for a $2 \times 2 \times 2$ cell

Method	Iterations	Total Frequencies	Degeneracy	Mistakes
NDPLE	5,5	2×10^5	-	281
	10,10	9.8×10^6	-	1.7×10^3
DPLE	5,5			
	10,10			

Table C.4: Mistakes for a $3 \times 3 \times 3$ cell

Appendix D

The Code

This appendix contains all the code used to calculate the density of states. In all the different methods used, the basic code was the same and changes had to be made in a few of the subroutines. We have included the code used for the non-perturbative method completely (since that one is the most effective one), and have included only the changes for the rest of the methods. It also includes the code used in the Embedded Atom Method.

Bibliography

[1] "I. M. Lifshitz, Nuovo Cim., 3 Suppl., 716 (1956)."

[2] "P. G. Dawber and R. J. Elliot, Proc. Roy. Soc. A, 81, 483 (1963)."

[3] "A. A. Maradudin, P. A. Flinn and S. Ruby, Phys. Rev., 126, 9 (1962)."

[4] "J. M. Rowe, J. J. Rush, H. G. Smith, Mark Mostoller and H. E. Flotow, Phys. Rev. Lett., 33, 1297 (1974)."

[5] "A. Rahman, K. Skold, C. Perizzari, S. K. Sinha and H. Flotow, Phys. Rev. B., 14, 3630 (1976)."

[6] "Mo Li and William A. Goddard III, Phys. Rev. B., 40, 12155 (1989)."

[7] "C. J. Glinka, J. M. Rowe, J. J. Rush, A. Rahman, S. K. Sinha, and H. E. Flotow, Phys. Rev. B, 17, 488 (1978)."

[8] "P. L. Hagelstein, Fusion Tech., 23 353 (1993)."

[9] "P. L. Hagelstein and S. Kaushik, Proc. ICCF4, 1, 10-1 (1994)."

[10] "Max Born and Kerson Huang, *Dynamical Theory of Crystal Lattices, Oxford University Press, London, 1956."*

[11] "M. S. Daw and M. I. Baskes, Phys. Rev. B 29, 6443 (1984)."

[12] "S. M. Foiles, M.I. Baskes, and M. S. Daw, Phys. Rev. B 33, 7983 (1986)."

[13] "J. J. Sakuarai, Modern Quantum Mechanics, Addison Wesley, U.S.A., 1985."

[14] "E. W. Kellerman, Proc. Royal Soc. London, 238, 513 (1940)."

[15] "L. A. Girifalco and Weizer, J. Phys. Chem. Solids, 12, 260 (1960)."

[16] "G. Gilat and G. Dolling, Phys. Lett., 8, 304 (1964)."

[17] "P. Hohenberg and W. Kohn, Phys. Rev., 136, B864 (1964)."

[18] "J. K. Norskov and N. D. Lang, Phys. Rev. B, 21, 2131 (1980)."

```
C  Embedding energy from from Pushka et al.
C  Pair potential energy from Murray and Dawes.
C
d  B1 = 20.44510
d  B2 = -2.897859
d  B3 = 52.89785
d  B4 = 0.412562
d  p  = 3.83
```

```
C     Degenerate Perturbation Theory

C     This is the command used to compile it on athena
C     f77 DP.f -L/mit/lapack/sun4lib -llapack -lblas

C     The lattice constant is 2 so that the difference between
C     adjacent atoms is only 1

      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

C     declare parameters
      double precision INIT,FINAL,STEP,LIMIT,NSTEPS
      double precision finit,ffinal,fstep,fnsteps

      PARAMETER (NSTEPS = 5.0)
      PARAMETER (INIT = 1.0/(2.0*NSTEPS))
      PARAMETER (FINAL = 1.0)
      PARAMETER (STEP = 1.0/NSTEPS)
      PARAMETER (LIMIT = 1.5 - INIT)

      PARAMETER (fnsteps = 5.0)
      PARAMETER (finit = -INIT)
      PARAMETER (fstep = (2.0*INIT)/fnsteps)
      PARAMETER (ffinal = INIT - fstep)

C     declare local variables
      integer i,j,IntOmega,MaxOmega
      double precision Mistakes,TotEigValue
      double precision QX,QY,QZ,fqx,fqy,fqz,omega,histogram(80)
      complex*16 DM(3*NASC,3*NASC),DeltaX(3*NASC,3*NASC)
      complex*16 TempDM(3*NASC,3*NASC),DMCopy(3*NASC,3*NASC)
      complex*16 DeltaXPrime(3*NASC,3*NASC)

C     declare variables for the lapack routine
      character JOBZ,UPLO
      integer INFO,LDA,LWORK,N
      double precision RWORK(3*(3*NASC)-2),W(3*NASC)
      complex*16 WORK(2*(3*NASC)-1)

C     define variables for checking degenracy in EigenValues
      integer DegValues,DegStart(3*NASC),DegNum(3*NASC)



C     initialize common variables

      a0 = 3.89

      l(1,1) = 2
      l(1,2) = 2
      l(1,3) = 0

      l(2,1) = 2
      l(2,2) = 0
      l(2,3) = 2

      l(3,1) = 0
      l(3,2) = 2
      l(3,3) = 2

      TDegenerate = 0.0
C     initialize variables for lapack routine

      JOBZ = 'V'
```

```
      UPLO = 'U'
      N = 3*NASC
      LDA = 3*NASC
      LWORK = 2*(3*NASC)-1

      do 5 j=1,80
         histogram(j) = 0.0
5     continue

      MaxOmega = 0
      Mistakes = 0.0
      TotEigValue = 0.0
      call LoadData()
      call NNeighbour()
      do 10 QX = INIT,FINAL,STEP
         print *,QX,TDegenerate
         do 20 QY = INIT,FINAL,STEP
            do 30 QZ = INIT,FINAL,STEP
               if (QX+QY+QZ .gt. LIMIT) then
                  continue
               elseif (QY .gt. QX) then
                  continue
               elseif (QZ .gt. QY) then
                  continue
               else
                  call DynMat(QX,QY,QZ,DM)
                  do 40 i=1,3*NASC
                     do 50 j=1,3*NASC
                        DMCopy(i,j) = DM(i,j)
50                   continue
40                continue

                  call zheev(JOBZ,UPLO,N,DM,LDA,W,WORK,LWORK,RWORK,INFO)
                  call CheckDeg(W,DegValues,DegStart,DegNum)

                  do 60 fqx=finit,ffinal,fstep
                     do 70 fqy=finit,ffinal,fstep
                        do 80 fqz=finit,ffinal,fstep
                           if((QX+fqx+QY+fqy+QZ+fqz).gt.LIMIT)
     *                          then
                              continue
                           elseif (QY+fqy .gt. QX+fqx) then
                              continue
                           elseif (QZ+fqz .gt. QY+fqy) then
                              continue
                           else
                              TotEigValue = TotEigValue+dble(3*NASC)
                              call DynMat(QX+fqx,QY+fqy,QZ+fqz,TempDM)
                              call SubMat(TempDM,DMCopy,DeltaX)
                              call MultMatrix(DM,DeltaX,DeltaXPrime,
     *                             DegValues,DegStart,DegNum)
                               call DegPerturb(DegValues,DegStart,
     *                             DegNum,DeltaXPrime)

                              do 100 j=1,3*NASC
                                 omega = W(j) + dble(DeltaXPrime(j,j))
                                 omega = omega*1.0e24/1.66057
                                 if (omega.gt.0) then
                                    omega = dsqrt(omega)
                                    omega = omega/1.5193e12
                                 else
                                    Mistakes = Mistakes + 1
                                    omega = 0.0
                                 endif
                                 IntOmega = nint(omega)
                                 if (IntOmega.gt.MaxOmega) then
```

```
                                    MaxOmega = IntOmega
                                 endif
                                 if (IntOmega.gt.0) then
                                    histogram(IntOmega) =
     *                                   histogram(IntOmega) + 1.0
                                 endif
 100                          continue
                           endif
 80                     continue
 70                  continue
 60               continue
               endif
 30         continue
 20      continue
 10    continue


       print *,TotEigValue,TDegenerate,Mistakes
       open (unit = 7,file = 'temp',status = 'unknown')
       do 500 j=1,MaxOmega
          write(7,*) histogram(j)
 500   continue
       close (unit = 7)
       end

C      ---------------------------------------------------------------
C      this subroutine does degenerate perturbation theory

       subroutine DegPerturb(DegValues,DegStart,DegNum,DeltaXPrime)
       include "hfparamPert.com"
       include "hfcomPert.dec"
       include "hfcomPert.def"

       integer DegValues,DegStart(3*NASC),DegNum(3*NASC)
       complex*16 DeltaXPrime(3*NASC,3*NASC)

C      declare local variables
       integer i,j,alpha,beta
       complex*16 MatrixX(3*NASC,3*NASC)


C      for the lapack routine
       character JOBZ,INFO
       double precision RWORK(3*(3*NASC)-2)
       double precision DegWx(3*NASC)
       complex*16 WORK(2*(3*NASC)-1)

       JOBZ = 'N'
       UPLO = 'U'
       LDA = 3*NASC
       LWORK = 2*(3*NASC)-1

       do 10 i=1,DegValues
          do 20 alpha=1,DegNum(i)+1
             do 30 beta=1,DegNum(i)+1
                MatrixX(alpha,beta) = DeltaXPrime(alpha+
     *               DegStart(i)-1,beta+DegStart(i)-1)
 30          continue
 20       continue

          call zheev(JOBZ,UPLO,DegNum(i)+1,MatrixX,
     *         LDA,DegWx,WORK,LWORK,RWORK,INFO)

          do 40 j=0,DegNum(i)
             DeltaXPrime(DegStart(i)+j,DegStart(i)+j)
     *            = DegWx(j+1)
```

```
 40      continue
 10   continue
      return
      end


C     ---------------------------------------------------------------
C     this subroutine finds degenerate eigenvalues.w is the eigenvector
C     DegValues contains the no. of degenerate eigenvalues, DegStart
C     tells the starting point of the degenerate eigenvalues, DegNum
C     gives the no. of each type of degenerate eigenvalue.e.g. if
C     v = [1 2 2 2 3 4 4 5 5 5] then DegValues = 3;DegStart(1) = 2
C     DegStart(2) = 6 DegStart(3) = 8. DegNum(1) = 2 DegNum(2) = 1
C     DegNum(3) = 2

      subroutine CheckDeg(w,DegValues,DegStart,DegNum)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

      double precision w(3*NASC)
      integer DegValues,DegStart(3*NASC),DegNum(3*NASC)

      integer i,j, counter
      double precision prev,current,Tol

      do 5 i=1,3*NASC
         DegStart(i) = 0
         DegNum(i) = 0
 5    continue
      Tol = 5
      prev = -18989.5454
      DegValues= 0
      counter = 0
      do 10 i=1,3*NASC-1
         current = w(i)
         if (abs(w(i+1)-current).lt.Tol .and. counter.eq.0)then
            DegValues = DegValues + 1
            DegStart(DegValues) = i
            do 20 j=i+1,3*NASC
               if (abs(w(j)-current).lt.Tol) then
                  DegNum(DegValues)= DegNum(DegValues)+1
                  counter = counter + 1
                  current = w(j)
               else
                  goto 10
               endif
 20         continue
         else
            if (counter.gt.0) then
               counter = counter - 1
            endif
         endif
 10   continue
      TDegenerate = TDegenerate + dble(DegValues)
      return
      end

C     ---------------------------------------------------------------
C     this subroutine subtracts DM from DMPert, and leaves the answer
C     in Delta

      subroutine SubMat(DMPert,DM,Delta)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"
```

```
      complex*16 DMPert(3*NASC,3*NASC),DM(3*NASC,3*NASC)
      complex*16 Delta(3*NASC,3*NASC)

      integer i,j
      do 10 i=1,3*NASC
         do 20 j=1,3*NASC
            Delta(i,j) = DMPert(i,j) - DM(i,j)
 20      continue
 10   continue
      return
      end

C     ----------------------------------------------------------------
C     this subroutine calculates DMdagger*DeltaX(Y or Z)*DM, and leaves
C     the result in DeltaX(Y or Z)Prime
      subroutine MultMatrix(DM,Delta,DeltaPrime,DegValues,DegStart,
     *     DegNum)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

      complex*16 DM(3*NASC,3*NASC),Delta(3*NASC,3*NASC)
      complex*16 DeltaPrime(3*NASC,3*NASC)
      integer DegValues,DegStart(3*NASC),DegNum(3*NASC)

C     declare local variables
      complex*16 DMdagger(3*NASC,3*NASC),temp
      integer i,j,alpha,beta,psi

      do 10 i=1,3*NASC
         do 20 j=1,3*NASC
            DeltaPrime(i,j) = (0.0,0.0)
            DMdagger(i,j) = dconjg(DM(j,i))
 20      continue
 10   continue

      do 30 alpha=1,3*NASC
         temp = (0.0,0.0)
         do 50 i=1,3*NASC
            do 60 j=1,3*NASC
               temp = temp + DMdagger(alpha,i)*Delta(i,j)*DM(j,alpha)
 60         continue
 50      continue
         DeltaPrime(alpha,alpha) = temp
 30   continue


      if (DegValues.gt.0) then
         do 70 psi=1,DegValues
            do 80 alpha = DegStart(psi),DegStart(psi)+DegNum(psi)
               do 90 beta = DegStart(psi),DegStart(psi)+DegNum(psi)
                  temp = (0.0,0.0)
                  do 100 i=1,3*NASC
                     do 110 j=1,3*NASC
                        temp = temp + DMdagger(alpha,i)*Delta(i,j)*
     *                       DM(j,beta)
 110                 continue
 100                 continue
                     DeltaPrime(alpha,beta) = temp
 90                  continue
 80             continue

 70      continue
      endif
      return
```

```
end
```

```
C     Degenerate Perturbation Theory With Linear Extrapolation

C     This is the command used to compile it on athena
C     f77 DPLE.f -L/mit/lapack/sun4lib -llapack -lblas

C     The lattice constant is 2 so that the difference between
C     adjacent atoms is only 1

      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

C     declare parameters
      double precision INIT,FINAL,STEP,LIMIT,NSTEPS
      double precision finit,ffinal,fstep,fnsteps

      PARAMETER (NSTEPS = 5.0)
      PARAMETER (INIT = 1.0/(2.0*NSTEPS))
      PARAMETER (FINAL = 1.0)
      PARAMETER (STEP = 1.0/NSTEPS)
      PARAMETER (LIMIT = 1.5 - INIT)

      PARAMETER (fnsteps = 5.0)
      PARAMETER (finit = -INIT)
      PARAMETER (fstep = (2.0*INIT)/fnsteps)
      PARAMETER (ffinal = INIT - fstep)

C     declare local variables
      integer j,IntOmega,MaxOmega
      double precision TotEigValues,Mistakes
      double precision QX,QY,QZ,fqx,fqy,fqz,omega,histogram(80)
      complex*16 DM(3*NASC,3*NASC),DeltaX(3*NASC,3*NASC)
      complex*16 DeltaY(3*NASC,3*NASC),DeltaZ(3*NASC,3*NASC)
      complex*16 TempDM(3*NASC,3*NASC)
      complex*16 DeltaXPrime(3*NASC,3*NASC)
      complex*16 DeltaYPrime(3*NASC,3*NASC)
      complex*16 DeltaZPrime(3*NASC,3*NASC)

C     declare variables for the lapack routine
      character JOBZ,UPLO
      integer INFO,LDA,LWORK,N
      double precision RWORK(3*(3*NASC)-2),W(3*NASC)
      complex*16 WORK(2*(3*NASC)-1)

C     define variables for checking degenracy in EigenValues
      integer DegValues,DegStart(3*NASC),DegNum(3*NASC)



C     initialize common variables

      a0 = 3.89

      l(1,1) = 3
      l(1,2) = 3
      l(1,3) = 0

      l(2,1) = 3
      l(2,2) = 0
      l(2,3) = 3

      l(3,1) = 0
      l(3,2) = 3
      l(3,3) = 3

      TDegenerate = 0.0
```

```
C     initialize variables for lapack routine

      JOBZ = 'V'
      UPLO = 'U'
      N = 3*NASC
      LDA = 3*NASC
      LWORK = 2*(3*NASC)-1


      TotEigValues = 0.0
      Mistakes = 0.0

      do 5 j=1,80
         histogram(j) = 0.0
 5    continue

      MaxOmega = 0

      call LoadData()
      call NNeighbour()
      do 10 QX = INIT,FINAL,STEP
         print *,QX
         do 20 QY = INIT,FINAL,STEP
            do 30 QZ = INIT,FINAL,STEP
               if (QX+QY+QZ .gt. LIMIT) then
                  continue
               elseif (QY .gt. QX) then
                  continue
               elseif (QZ .gt. QY) then
                  continue
               else
                  call DynMat(QX,QY,QZ,DM)
                  call DynMat(QX+fstep,QY,QZ,TempDM)
                  call SubMat(TempDM,DM,DeltaX)
                  call DynMat(QX,QY+fstep,QZ,TempDM)
                  call SubMat(TempDM,DM,DeltaY)
                  call DynMat(QX,QY,QZ+fstep,TempDM)
                  call SubMat(TempDM,DM,DeltaZ)
                  call zheev(JOBZ,UPLO,N,DM,LDA,W,WORK,LWORK,RWORK,INFO)
                  call CheckDeg(W,DegValues,DegStart,DegNum)
                  call MultMatrix(DM,DeltaX,DeltaXPrime,DegValues,
     *                 DegStart,DegNum)
                  call MultMatrix(DM,DeltaY,DeltaYPrime,DegValues,
     *                 DegStart,DegNum)
                  call MultMatrix(DM,DeltaZ,DeltaZPrime,DegValues,
     *                 DegStart,DegNum)
                  call DegPerturb(DegValues,DegStart,DegNum,
     *                 DeltaXPrime,DeltaYPrime,DeltaZPrime)

                  do 60 fqx=finit,ffinal,fstep
                     do 70 fqy=finit,ffinal,fstep
                        do 80 fqz=finit,ffinal,fstep
                           if((QX+fqx+QY+fqy+QZ+fqz).gt.LIMIT)
     *                          then
                              continue
                           elseif (QY+fqy .gt. QX+fqx) then
                              continue
                           elseif (QZ+fqz .gt. QY+fqy) then
                              continue
                           else
                              TotEigValues = TotEigValues+dble(3*NASC)
                              do 100 j=1,3*NASC
                                 omega = W(j) + dble(
     *                                (DeltaXPrime(j,j)*fqx
     *                                + DeltaYPrime(j,j)*fqy +
```

```
     *                                DeltaZPrime(j,j)*fqz))/fstep

                                    omega = omega*1.0e24/1.66057
                                    if (omega.gt.0) then
                                       omega = dsqrt(omega)
                                       omega = omega/1.5193e12
                                    else
                                       Mistakes = Mistakes + 1.0
                                       omega = 0.0
                                    endif
                                    IntOmega = nint(omega)
                                    if (IntOmega.gt.MaxOmega) then
                                       MaxOmega = IntOmega
                                    endif
                                    if (IntOmega.gt.0) then
                                       histogram(IntOmega) =
     *                                      histogram(IntOmega) + 1.0
                                    endif
 100                             continue
                              endif
  80                        continue
  70                    continue
  60                continue
                    endif
  30            continue
  20        continue
  10    continue


        print 200,TotEigValues,TDegenerate,Mistakes
        open (unit = 7,file = 'temp',status = 'unknown')
        do 500 j=1,MaxOmega
           write(7,*) histogram(j)
  500   continue
        close (unit = 7)
  200   format(e16.8,e16.8)
        end

C       ---------------------------------------------------------------
C       this subroutine does degenerate perturbation theory

        subroutine DegPerturb(DegValues,DegStart,DegNum,DeltaXPrime,
       *     DeltaYPrime,DeltaZPrime)
        include "hfparamPert.com"
        include "hfcomPert.dec"
        include "hfcomPert.def"

        integer DegValues,DegStart(3*NASC),DegNum(3*NASC)
        complex*16 DeltaXPrime(3*NASC,3*NASC),DeltaYPrime(3*NASC,3*NASC)
        complex*16 DeltaZPrime(3*NASC,3*NASC)

C       declare local variables
        integer i,j,alpha,beta
        complex*16 MatrixX(3*NASC,3*NASC),MatrixY(3*NASC,3*NASC)
        complex*16 MatrixZ(3*NASC,3*NASC)

C       for the lapack routine
        character JOBZ,INFO
        double precision RWORK(3*(3*NASC)-2)
        double precision DegWx(3*NASC),DegWy(3*NASC),DegWz(3*NASC)
        complex*16 WORK(2*(3*NASC)-1)

        JOBZ = 'N'
        UPLO = 'U'
        LDA = 3*NASC
        LWORK = 2*(3*NASC)-1
```

```
      do 10 i=1,DegValues
         do 20 alpha=1,DegNum(i)+1
            do 30 beta=1,DegNum(i)+1
               MatrixX(alpha,beta) = DeltaXPrime(alpha+
     *              DegStart(i)-1,beta+DegStart(i)-1)
               MatrixY(alpha,beta) = DeltaYPrime(alpha+
     *              DegStart(i)-1,beta+DegStart(i)-1)
               MatrixZ(alpha,beta) = DeltaZPrime(alpha+
     *              DegStart(i)-1,beta+DegStart(i)-1)
 30         continue
 20      continue

         call zheev(JOBZ,UPLO,DegNum(i)+1,MatrixX,
     *        LDA,DegWx,WORK,LWORK,RWORK,INFO)
         call zheev(JOBZ,UPLO,DegNum(i)+1,MatrixY,
     *        LDA,DegWy,WORK,LWORK,RWORK,INFO)
         call zheev(JOBZ,UPLO,DegNum(i)+1,MatrixZ,
     *        LDA,DegWz,WORK,LWORK,RWORK,INFO)

         do 40 j=0,DegNum(i)
            DeltaXPrime(DegStart(i)+j,DegStart(i)+j)
     *           = DegWx(j+1)
            DeltaYPrime(DegStart(i)+j,DegStart(i)+j)
     *           = DegWy(j+1)
            DeltaZPrime(DegStart(i)+j,DegStart(i)+j)
     *           = DegWz(j+1)

 40      continue
 10   continue
      return
      end


C     ---------------------------------------------------------------
C     this subroutine finds degenerate eigenvalues.w is the eigenvector
C     DegValues contains the no. of degenerate eigenvalues, DegStart
C     tells the starting point of the degenerate eigenvalues, DegNum
C     gives the no. of each type of degenerate eigenvalue.e.g. if
C     v = [1 2 2 2 3 4 4 5 5 5] then DegValues = 3;DegStart(1) = 2
C     DegStart(2) = 6 DegStart(3) = 8. DegNum(1) = 2 DegNum(2) = 1
C     DegNum(3) = 2

      subroutine CheckDeg(w,DegValues,DegStart,DegNum)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

      double precision w(3*NASC)
      integer DegValues,DegStart(3*NASC),DegNum(3*NASC)

      integer i,j, counter
      double precision prev,current,Tol

      do 5 i=1,3*NASC
         DegStart(i) = 0
         DegNum(i) = 0
 5    continue
      Tol = 5
      prev = -1800.5454
      DegValues= 0
      counter = 0
      do 10 i=1,3*NASC-1
         current = w(i)
         if (abs(w(i+1)-current).lt.Tol .and. counter.eq.0)then
            DegValues = DegValues + 1
```

```
            DegStart(DegValues) = i
            do 20 j=i+1,3*NASC
               if (abs(w(j)-current).lt.Tol) then
                  DegNum(DegValues)= DegNum(DegValues)+1
                  counter = counter + 1
                  current = w(j)
               else
                  goto 10
               endif
 20         continue
         else
            if (counter.gt.0) then
               counter = counter - 1
            endif
         endif
 10   continue
      TDegenerate = TDegenerate + dble(DegValues)
      return
      end

C     ----------------------------------------------------------------
C     this subroutine subtracts DM from TempDM, and leaves the answer
C     in Delta

      subroutine SubMat(TempDM,DM,Delta)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

      complex*16 TempDM(3*NASC,3*NASC),DM(3*NASC,3*NASC)
      complex*16 Delta(3*NASC,3*NASC)

      integer i,j
      do 10 i=1,3*NASC
         do 20 j=1,3*NASC
            Delta(i,j) = TempDM(i,j) - DM(i,j)
 20      continue
 10   continue
      return
      end

C     ----------------------------------------------------------------
C     this subroutine calculates DMdagger*DeltaX(Y or Z)*DM, and leaves
C     the result in DeltaX(Y or Z)Prime
      subroutine MultMatrix(DM,Delta,DeltaPrime,DegValues,DegStart,
     *     DegNum)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

      complex*16 DM(3*NASC,3*NASC),Delta(3*NASC,3*NASC)
      complex*16 DeltaPrime(3*NASC,3*NASC)
      integer DegValues,DegStart(3*NASC),DegNum(3*NASC)

C     declare local variables
      complex*16 DMdagger(3*NASC,3*NASC),temp,temp1,temp2
      integer i,j,alpha,beta,psi

      do 10 i=1,3*NASC
         do 20 j=1,3*NASC
            DeltaPrime(i,j) = (0.0,0.0)
            DMdagger(i,j) = dconjg(DM(j,i))
 20      continue
 10   continue

      do 30 alpha=1,3*NASC
```

```
      temp = (0.0,0.0)
      temp1 = (0.0,0.0)
      temp2 = (0.0,0.0)
      do 50 i=1,3*NASC
         do 60 j=1,3*NASC
            temp = temp + DMdagger(alpha,i)*Delta(i,j)*DM(j,alpha)
            if ((alpha-1).gt.0) then
               temp1 = temp1 + DMdagger(alpha,i)*Delta(i,j)
     *              *DM(j,alpha-1)
            endif
            if((alpha+1).lt.(3*NASC)) then
               temp2 = temp2 + DMdagger(alpha,i)*Delta(i,j)*
     *              DM(j,alpha+1)
            endif
60       continue
50    continue
      if ((alpha-1).gt.0) then
         DeltaPrime(alpha,alpha-1) = temp1
         DeltaPrime(alpha-1,alpha) = dconjg(temp1)
      endif
      if ((alpha+1).lt.(3*NASC)) then
         DeltaPrime(alpha,alpha+1) = temp2
         DeltaPrime(alpha+1,alpha) = dconjg(temp2)
      endif
      DeltaPrime(alpha,alpha) = temp
30   continue


     if (DegValues.gt.0) then
        do 70 psi=1,DegValues
           do 80 alpha = DegStart(psi),DegStart(psi)+DegNum(psi)
              do 90 beta = DegStart(psi),DegStart(psi)+DegNum(psi)
                 if (DeltaPrime(alpha,beta).eq.(0.0,0.0)) then
                    temp = (0.0,0.0)
                    do 100 i=1,3*NASC
                       do 110 j=1,3*NASC
                          temp = temp + DMdagger(alpha,i)*Delta(i,j)*
     *                         DM(j,beta)
110                    continue
100                 continue
                    DeltaPrime(alpha,beta) = temp
                    DeltaPrime(beta,alpha) = dconjg(temp)
                 endif
90            continue
80         continue

70      continue
     endif
     return
     end
```

```
C     Non-Degenerate Perturbation Theory

C     This is the command used to compile it on athena
C     f77 NDP.f -L/mit/lapack/sun4lib -llapack -lblas

C     The lattice constant is 2 so that the difference between
C     adjacent atoms is only 1

      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

C     declare parameters
      double precision INIT,FINAL,STEP,LIMIT,NSTEPS
      double precision finit,ffinal,fstep,fnsteps

      PARAMETER (NSTEPS = 5.0)
      PARAMETER (INIT = 1.0/(2.0*NSTEPS))
      PARAMETER (FINAL = 1.0)
      PARAMETER (STEP = 1.0/NSTEPS)
      PARAMETER (LIMIT = 1.5 - INIT)

      PARAMETER (fnsteps = 5.0)
      PARAMETER (finit = -INIT)
      PARAMETER (fstep = (2.0*INIT)/fnsteps)
      PARAMETER (ffinal = INIT - fstep)


C     declare local variables
      integer j,IntOmega,MaxOmega,Mistakes
      double precision QX,QY,QZ,fqx,fqy,fqz,omega,histogram(80)
      complex*16 DM(3*NASC,3*NASC),DeltaX(3*NASC,3*NASC)
      complex*16 DMPert(3*NASC,3*NASC)
      complex*16 DeltaXPrime(3*NASC,3*NASC)
      complex*16,DMCopy(3*NASC,3*NASC)

C     declare variables for the lapack routine
      character JOBZ,UPLO
      integer INFO,LDA,LWORK,N
      double precision RWORK(3*(3*NASC)-2),W(3*NASC)
      complex*16 WORK(2*(3*NASC)-1)

C     initialize common variables

      a0 = 3.89

      l(1,1) = 2
      l(1,2) = 2
      l(1,3) = 0

      l(2,1) = 2
      l(2,2) = 0
      l(2,3) = 2

      l(3,1) = 0
      l(3,2) = 2
      l(3,3) = 2

C     initialize variables for lapack routine

      JOBZ = 'V'
      UPLO = 'U'
      N = 3*NASC
      LDA = 3*NASC
      LWORK = 2*(3*NASC)-1
```

```
      do 5 j=1,80
         histogram(j) = 0.0
5     continue

      MaxOmega = 0
      Mistakes = 0

      call LoadData()
      call NNeighbour()

      do 10 QX=INIT,FINAL,STEP
         print *,QX
         do 20 QY = INIT,FINAL,STEP
            do 30 QZ = INIT,FINAL,STEP
               if (QX+QY+QZ .gt. LIMIT) then
                  continue
               elseif (QY .gt. QX) then
                  continue
               elseif (QZ .gt. QY) then
                  continue
               else
                  call DynMat(QX,QY,QZ,DM)
                  do 40 i=1,3*NASC
                     do 50 j=1,3*NASC
                        DMCopy(i,j) = DM(i,j)
50                   continue
40                 continue
                   call zheev(JOBZ,UPLO,N,DM,LDA,W,WORK,LWORK,RWORK,INFO)
                   do 60 fqx=finit,ffinal,fstep
                      do 70 fqy=finit,ffinal,fstep
                         do 80 fqz=finit,ffinal,fstep
                            if((QX+fqx+QY+fqy+QZ+fqz).gt.LIMIT)
     *                           then
                               continue
                            elseif (QY+fqy .gt. QX+fqx) then
                               continue
                            elseif (QZ+fqz .gt. QY+fqy) then
                               continue
                            else
                               call DynMat(QX+fqx,QY+fqy,QZ+fqz,DMPert)
                               call SubMat(DMPert,DMCopy,DeltaX)
                               call MultMatrix(DM,DeltaX,DeltaXPrime)
                               do 100 j=1,3*NASC
                                  omega = W(j) + dble(DeltaXPrime(j,j))
                                  omega = omega*1.0e24/1.66057
                                  if(omega.gt.0) then
                                     omega = sqrt(omega)
                                     omega = omega/1.5193e12
                                  else
                                     omega = 0.0
                                     Mistakes = Mistakes + 1
                                  endif
                                  IntOmega = nint(Omega)
                                  if (IntOmega.gt.MaxOmega) then
                                     Maxomega = IntOmega
                                  endif
                                  if (IntOmega.gt.0) then
                                     histogram(IntOmega)=
     *                                    histogram(IntOmega) + 1.0
                                  endif
100                            continue
                            endif
80                       continue
70                    continue
60                 continue
               endif
```

```
 30         continue
 20      continue
 10   continue

      print *,Mistakes
      open (unit = 7,file = 'temp',status = 'unknown')
      do 200 j=1,MaxOmega
         write(7,*) histogram(j)
 200  continue
      close (unit = 7)
      end

C     ---------------------------------------------------------------
C     this subroutine subtracts DM from DMPert, and leaves the answer
C     in DeltaX

      subroutine SubMat(DMPert,DM,Delta)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

      complex*16 DMPert(3*NASC,3*NASC),DM(3*NASC,3*NASC)
      complex*16 Delta(3*NASC,3*NASC)

      integer i,j
      do 10 i=1,3*NASC
         do 20 j=1,3*NASC
            Delta(i,j) = DMPert(i,j) - DM(i,j)
 20      continue
 10   continue
      return
      end

C     ---------------------------------------------------------------
C     this subroutine calculates DMdagger*DeltaX(Y or Z)*DM, and leaves
C     the result in DeltaX(Y or Z)Prime.
C     NOTE: only the diagonals of DeltaPrime are calculated

      subroutine MultMatrix(DM,Delta,DeltaPrime)
      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

      complex*16 DM(3*NASC,3*NASC),Delta(3*NASC,3*NASC)
      complex*16 DeltaPrime(3*NASC,3*NASC)

      complex*16 DMdagger(3*NASC,3*NASC),temp
      integer i,j,alpha

      do 10 i=1,3*NASC
         do 20 j=1,3*NASC
            DeltaPrime(i,j) = (0.0,0.0)
            DMdagger(i,j) = dconjg(DM(j,i))
 20      continue
 10   continue


      do 30 alpha=1,3*NASC
         temp = (0.0,0.0)
         do 50 i=1,3*NASC
            do 60 j=1,3*NASC
               temp=temp+DMdagger(alpha,i)*Delta(i,j)*DM(j,alpha)
 60         continue
 50      continue
         DeltaPrime(alpha,alpha) = temp
 30   continue
```

```
return
end
```

```
C     Non-Degenerate Perturbation Theory With Linear Extrapolation

C     This is the command used to compile it on athena
C     f77 NDPLE.f -L/mit/lapack/sun4lib -llapack -lblas

C     The lattice constant is 2 so that the difference between
C     adjacent atoms is only 1

      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

C     declare parameters
      double precision INIT,FINAL,STEP,LIMIT,NSTEPS
      double precision finit,ffinal,fstep,fnsteps

      PARAMETER (NSTEPS = 20.0)
      PARAMETER (INIT = 1.0/(2.0*NSTEPS))
      PARAMETER (FINAL = 1.0)
      PARAMETER (STEP = 1.0/NSTEPS)
      PARAMETER (LIMIT = 1.5 - INIT)

      PARAMETER (fnsteps = 20.0)
      PARAMETER (finit = -INIT)
      PARAMETER (fstep = (2.0*INIT)/fnsteps)
      PARAMETER (ffinal = INIT - fstep)


C     declare local variables
      integer j,IntOmega,MaxOmega
      double precision TotEigValues,Mistakes
      double precision QX,QY,QZ,fqx,fqy,fqz,omega,histogram(80)
      complex*16 DM(3*NASC,3*NASC),DeltaX(3*NASC,3*NASC)
      complex*16 DeltaY(3*NASC,3*NASC),DeltaZ(3*NASC,3*NASC)
      complex*16 DMPert(3*NASC,3*NASC)
      complex*16 DeltaXPrime(3*NASC,3*NASC)
      complex*16 DeltaYPrime(3*NASC,3*NASC)
      complex*16 DeltaZPrime(3*NASC,3*NASC)

C     declare variables for the lapack routine
      character JOBZ,UPLO
      integer INFO,LDA,LWORK,N
      double precision RWORK(3*(3*NASC)-2),W(3*NASC)
      complex*16 WORK(2*(3*NASC)-1)

C     initialize common variables

      a0 = 3.89

      l(1,1) = 2
      l(1,2) = 2
      l(1,3) = 0

      l(2,1) = 2
      l(2,2) = 0
      l(2,3) = 2

      l(3,1) = 0
      l(3,2) = 2
      l(3,3) = 2

C     initialize variables for lapack routine

      JOBZ = 'V'
      UPLO = 'U'
      N = 3*NASC
```

```
      LDA = 3*NASC
      LWORK = 2*(3*NASC)-1

      do 5 j=1,80
         histogram(j) = 0.0
5     continue

      MaxOmega = 0
      Mistakes = 0.0
      TotEigValues = 0.0

      call LoadData()
      call NNeighbour()

      do 10 QX=INIT,FINAL,STEP
         print *,QX
         do 20 QY = INIT,FINAL,STEP
            do 30 QZ = INIT,FINAL,STEP
               if (QX+QY+QZ .gt. LIMIT) then
                  continue
               elseif (QY .gt. QX) then
                  continue
               elseif (QZ .gt. QY) then
                  continue
               else
                  call DynMat(QX,QY,QZ,DM)
                  call DynMat(QX+fstep,QY,QZ,DMPert)
                  call SubMat(DMPert,DM,DeltaX)
                  call DynMat(QX,QY+fstep,QZ,DMPert)
                  call SubMat(DMPert,DM,DeltaY)
                  call DynMat(QX,QY,QZ+fstep,DMPert)
                  call SubMat(DMPert,DM,DeltaZ)

                  call zheev(JOBZ,UPLO,N,DM,LDA,W,WORK,
     *                 LWORK,RWORK,INFO)

                  call MultMatrix(DM,DeltaX,DeltaXPrime)
                  call MultMatrix(DM,DeltaY,DeltaYPrime)
                  call MultMatrix(DM,DeltaZ,DeltaZPrime)

                  do 40 fqx=finit,ffinal,fstep
                     do 50 fqy=finit,ffinal,fstep
                        do 60 fqz=finit,ffinal,fstep
                           if((QX+fqx+QY+fqy+QZ+fqz).gt.LIMIT)
     *                          then
                              continue
                           elseif (QY+fqy .gt. QX+fqx) then
                              continue
                           elseif (QZ+fqz .gt. QY+fqy) then
                              continue
                           else
                              TotEigValues = TotEigValues+dble(3*NASC)
                              do 100 j=1,3*NASC
                                 omega = W(j)+dble(
     *                                (DeltaXPrime(j,j)*fqx+
     *                                DeltaYPrime(j,j)*fqy +
     *                                DeltaZPrime(j,j)*fqz))/fstep
                                 omega = omega*1.0e24/1.66057
                                 if(omega.gt.0) then
                                    omega = sqrt(omega)
                                    omega = omega/1.5193e12
                                 else
                                    omega = 0.0
                                    Mistakes = Mistakes + 1.0
                                 endif
                                 IntOmega = nint(omega)
```

```
                                    if (IntOmega.gt.MaxOmega) then
                                       Maxomega = IntOmega
                                    endif
                                    if (IntOmega.gt.0) then
                                       histogram(IntOmega)=
     *                                      histogram(IntOmega) + 1.0
                                    endif
 100                             continue
                              endif
 60                        continue
 50                     continue
 40                  continue
                  endif
 30            continue
 20         continue
 10      continue

         print 200,TotEigValues,Mistakes
         open (unit = 7,file = 'temp',status = 'unknown')
         do 500 j=1,MaxOmega
            write(7,*) histogram(j)
 500     continue
         close (unit = 7)
 200     format(e16.8,e16.8)
         end

C        ---------------------------------------------------------------
C        this subroutine subtracts DM from DMPert, and leaves the answer
C        in DeltaX

         subroutine SubMat(DMPert,DM,Delta)
         include "hfparamPert.com"
         include "hfcomPert.dec"
         include "hfcomPert.def"

         complex*16 DMPert(3*NASC,3*NASC),DM(3*NASC,3*NASC)
         complex*16 Delta(3*NASC,3*NASC)

         integer i,j
         do 10 i=1,3*NASC
            do 20 j=1,3*NASC
               Delta(i,j) = DMPert(i,j) - DM(i,j)
 20         continue
 10      continue
         return
         end

C        ---------------------------------------------------------------
C        this subroutine calculates DMdagger*DeltaX(Y or Z)*DM, and leaves
C        the result in DeltaX(Y or Z)Prime.
C        NOTE: only the diagonals of DeltaPrime are calculated

         subroutine MultMatrix(DM,Delta,DeltaPrime)
         include "hfparamPert.com"
         include "hfcomPert.dec"
         include "hfcomPert.def"

         complex*16 DM(3*NASC,3*NASC),Delta(3*NASC,3*NASC)
         complex*16 DeltaPrime(3*NASC,3*NASC)

         complex*16 DMdagger(3*NASC,3*NASC),temp
         integer i,j,alpha

         do 10 i=1,3*NASC
            do 20 j=1,3*NASC
```

```
            DeltaPrime(i,j) = (0.0,0.0)
            DMdagger(i,j) = dconjg(DM(j,i))
20       continue
10    continue


      do 30 alpha=1,3*NASC
         temp = (0.0,0.0)
         do 50 i=1,3*NASC
            do 60 j=1,3*NASC
               temp=temp+DMdagger(alpha,i)*Delta(i,j)*DM(j,alpha)
60          continue
50       continue
         DeltaPrime(alpha,alpha) = temp
30    continue
      return
      end
```

```
C     No Perturbation.  This Is The Most Effective Way!

C     This is the command used to compile it on athena
C     f77 NoPert.f -L/mit/lapack/sun4lib -llapack -lblas

C     The lattice constant is 2 so that the difference between
C     adjacent atoms is only 1

      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

C     declare parameters
      double precision INIT,FINAL,STEP,LIMIT,NSTEPS
      integer RESOLUTION,HIGHFREQ,LOWFREQ,RESOLUTION2

      PARAMETER (NSTEPS = 20.0)
      PARAMETER (INIT = 1.0/(2.0*NSTEPS))
      PARAMETER (FINAL = 1.0)
      PARAMETER (STEP = 1.0/NSTEPS)
      PARAMETER (LIMIT = 1.5 - INIT)

      PARAMETER (LOWFREQ = 40.0)
      PARAMETER (HIGHFREQ = 60.0)
      PARAMETER (RESOLUTION = 100)

      PARAMETER (RESOLUTION2 = 10)

C     declare local variables
      integer j,k,IntOmega,MaxOmega,Mistakes
      double precision QX,QY,QZ,omega,histogram(80)
      double precision HistHighRes((HIGHFREQ-LOWFREQ)*RESOLUTION)
      double precision HistMedRes((HIGHFREQ-LOWFREQ)*RESOLUTION2)
      integer IntTempOmega
      double precision TempOmega
      complex*16 DM(3*NASC,3*NASC)

C     declare variables for the lapack routine
      character JOBZ,UPLO
      integer INFO,LDA,LWORK,N
      double precision RWORK(3*(3*NASC)-2),W(3*NASC)
      complex*16 WORK(2*(3*NASC)-1)

C     initialize common variables

      a0 = 3.89

      l(1,1) = 2
      l(1,2) = 2
      l(1,3) = 0

      l(2,1) = 2
      l(2,2) = 0
      l(2,3) = 2

      l(3,1) = 0
      l(3,2) = 2
      l(3,3) = 2

C     initialize variables for lapack routine

      JOBZ = 'N'
      UPLO = 'U'

      N = 3*NASC
      LDA = 3*NASC
```

```
      LWORK = 2*(3*NASC)-1

      do 5 j=1,80
         histogram(j) = 0.0
5     continue

      do 7 j=1,(HIGHFREQ-LOWFREQ)*RESOLUTION
         HistHighRes(j) = 0
7     continue

      do 8 j=1,(HIGHFREQ-LOWFREQ)*RESOLUTION2
         HistMedRes(j) = 0
8     continue

      MaxOmega = 0
      Mistakes = 0

      call LoadData()
      call NNeighbour()

      do 10 QX=INIT,FINAL,STEP
         print *,'QX'
         print *,QX
         do 20 QY = INIT,FINAL,STEP
            do 30 QZ = INIT,FINAL,STEP
               if (QX+QY+QZ .gt. LIMIT) then
                  continue
               elseif (QY .gt. QX) then
                  continue
               elseif (QZ .gt. QY) then
                  continue
               else
                  call DynMat(QX,QY,QZ,DM)
                  call zheev(JOBZ,UPLO,N,DM,LDA,W,WORK,LWORK,RWORK,INFO)
                  do 100 j=1,3*NASC
                     omega = W(j)
                     omega = omega*1.0e24/1.66057
                     if(omega.gt.0) then
                        omega = sqrt(omega)
                        omega = omega/1.5193e12
                     else
                        omega = 0.0
                        Mistakes = Mistakes + 1
                     endif

                     if ((omega.gt.LOWFREQ).and.(omega.lt.
     *                    HIGHFREQ)) then
                        TempOmega = (Omega - LOWFREQ) *
     *                       dble(RESOLUTION)
                        IntTempOmega = nint(TempOmega)
                        if (IntTempOmega.gt.0) then
                           HistHighRes(IntTempOmega) =
     *                          HistHighRes(IntTempOmega)
     *                          +1.0
                        endif
                        TempOmega = (Omega-LOWFREQ) *
     *                                    dble(RESOLUTION2)
                        IntTempOmega = nint(TempOmega)
                        if (IntTempOmega.gt.0) then
                           HistMedRes(IntTempOmega) =
     *                          HistMedRes(IntTempOmega)
     *                          + 1.0
                        endif
                     endif
```

```
                     IntOmega = nint(omega)
C                     if (IntOmega.eq.33) then
C                        print *,''
C                        print *,'33'
C                        do 40 k=1,3*NASC
C                           print *,DM(j,k),j,k
C 40                     continue
C                     endif
C                     if (IntOmega.eq.34) then
C                        print *,''
C                        print *,'34'
C                        do 50 k=1,3*NASC
C                           print *,DM(j,k),j,k
C 50                     continue
C                     endif
                     if (IntOmega.gt.MaxOmega) then
                        Maxomega = IntOmega
                     endif
                     if (IntOmega.gt.0) then
                        histogram(IntOmega)=histogram(IntOmega) + 1.0
                     endif
 100              continue
                endif
 30           continue
 20        continue
 10     continue

        print *,Mistakes
        open (unit = 7,file = 'temp',status = 'unknown')
        do 500 j=1,MaxOmega
           write(7,*) histogram(j)
 500    continue
        close (unit = 7)

        open (unit = 7,file = 'highres',status = 'unknown')
        do 600 j=1,(HIGHFREQ-LOWFREQ)*RESOLUTION
           write(7,*) HistHighRes(j)
 600    continue
        close (unit = 7)

        open (unit = 7,file = 'medres',status = 'unknown')
        do 700 j=1,(HIGHFREQ-LOWFREQ)*RESOLUTION2
           write (7,*) HistMedRes(j)
 700    continue
        close (unit = 8)


        end

C       ---------------------------------------------------------------

        subroutine DynMat(kx,ky,kz,DM)

        include "hfparamPert.com"
        include "hfcomPert.dec"
        include "hfcomPert.def"

        double precision kx,ky,kz
        complex*16 DM(3*NASC,3*NASC)

C       declare local variables

        integer i,j,a,b,x,y,TypeNN
        complex*16 sum,TempMat(3,3),TempDM(3*NASC,3*NASC)
        double precision KDotR,MassA,AM(3,3)
```

```
      do 5 i=1,3*NASC
         do 7 j=1,3*NASC
            DM(i,j) = (0.0,0.0)
            TempDM(i,j) = (0.0,0.0)
 7       continue
 5    continue

C     do the non-diagonal elements first

      do 10 i=1,NASC
         if (TypeSC(i) .eq. 46) then
            MassA = MPd
         else
            MassA = MD
         endif

         do 20 j=1,NSNN(i)
            call AppMat(AM,TypeSC(i),TypeSC(i),
     *           SNN(i,j,1),SNN(i,j,2),SNN(i,j,3))
            KDotR = (kx*SNN(i,j,1) + ky*SNN(i,j,2) + kz*SNN(i,j,3))*
     *           Pi
            do 30 a=1,3
               do 40 b=1,3
                  DM(3*(i-1)+a,3*(SList(i,j)-1)+b) =
     *                 DM(3*(i-1)+a,3*(SList(i,j)-1)+b) - AM(a,b)*
     *                 exp(iota*KDotR)/MassA
                  TempDM(3*(i-1)+a,3*(SList(i,j)-1)+b) =
     *                 TempDM(3*(i-1)+a,3*(SList(i,j)-1)+b) +
     *                 AM(a,b)/MassA

 40            continue
 30         continue
 20      continue

         if (TypeSC(i) .eq. 1) then
            MassA = MD
            TypeNN = 46
         else
            TypeNN = 1
            MassA = MPd
         endif

         do 50 j=1,NDNN(i)
            call AppMat(AM,TypeSC(i),TypeNN,
     *           DNN(i,j,1),DNN(i,j,2),DNN(i,j,3))
            KDotR = (kx*DNN(i,j,1) + ky*DNN(i,j,2) + kz*DNN(i,j,3))*
     *           Pi
            do 60 a=1,3
               do 70 b=1,3
                  DM(3*(i-1)+a,3*(DList(i,j)-1)+b) =
     *                 DM(3*(i-1)+a,3*(DList(i,j)-1)+b) - AM(a,b)*
     *                 exp(iota*KDotR)/sqrt(MPd*MD)
                  TempDM(3*(i-1)+a,3*(DList(i,j)-1)+b) =
     *                 TempDM(3*(i-1)+a,3*(DList(i,j)-1)+b) +
     *                 AM(a,b)/MassA

 70            continue
 60         continue
 50      continue

         if (TypeSC(i) .eq. 46) then
            MassA = MPd
         else
            MassA = MD
         endif
```

```
          do 80 j=1,NSSNN(i)
             call AppMat(AM,TypeSC(i),TypeSC(i),
     *            SSNN(i,j,1),SSNN(i,j,2),SSNN(i,j,3))
             KDotR = (kx*SSNN(i,j,1) + ky*SSNN(i,j,2) + kz*SSNN(i,j,3))*
     *            Pi
             do 90 a=1,3
                do 100 b=1,3
                   DM(3*(i-1)+a,3*(SList1(i,j)-1)+b) =
     *                 DM(3*(i-1)+a,3*(SList1(i,j)-1)+b) - AM(a,b)*
     *                 exp(iota*KDotR)/MassA
                   TempDM(3*(i-1)+a,3*(SList1(i,j)-1)+b) =
     *                 TempDM(3*(i-1)+a,3*(SList1(i,j)-1)+b) +
     *                 AM(a,b)/MassA
 100             continue
 90           continue
 80        continue
 10     continue

C       so now take care of the "diagonal" elements. Actually they are
C       3*3 matrices


       do 110 a=1,NASC*3,3
          do 120 i=1,3
             do 130 j=1,3
                sum = 0.0
                do 140 b=0,NASC*3 - 1,3
                   sum = sum + TempDM(a+i-1,b+j)
 140            continue
                TempMat(i,j) = sum
 130         continue
 120      continue
          do 150 x=1,3
             do 160 y=1,3
                DM(a+x-1,a+y-1) = DM(a+x-1,a+y-1) + TempMat(x,y)
 160         continue
 150      continue
 110   continue
       return
       end


C      ---------------------------------------------------------------
C      this subroutine calculates the nearest neighbours of the atoms
C      in the supercell

       subroutine NNeighbour()

       include "hfparamPert.com"
       include "hfcomPert.dec"
       include "hfcomPert.def"


C      declare local variables
       integer alpha,beta,i,j,k,TypeA,TypeNN
       double precision DistA1,DistA2,DistA3,SelfNNDist,DifDist,distance
       double precision DispX,DispY,DispZ,DispBeta(3)
       double precision SelfSNNDist

       DistA1 = sqrt(float(l(1,1)**2 + l(1,2)**2 + l(1,3)**2))
       DistA2 = sqrt(float(l(2,1)**2 + l(2,2)**2 + l(2,3)**2))
       DistA3 = sqrt(float(l(3,1)**2 + l(3,2)**2 + l(3,3)**2))
       SelfNNDist = min(DistA1,DistA2,DistA3)
       SelfSNNDist = 2*SelfNNDist
       DifDist = DistA1 + DistA2 + DistA3
       do 10 alpha=1,NASC
```

```
      TypeA = TypeSC(alpha)
      NSNN(alpha) = 0
      NDNN(alpha) = 0
      NSSNN(alpha) = 0
      do 20 beta=1,NASC
         TypeNN = TypeSC(beta)
         do 30 i=-2,2
            do 40 j=-2,2
               do 50 k=-2,2

                  DispX = i*l(1,1) + j*l(2,1) + k*l(3,1)
                  DispY = i*l(1,2) + j*l(2,2) + k*l(3,2)
                  DispZ = i*l(1,3) + j*l(2,3) + k*l(3,3)

                  DispBeta(1) = DispX + SuperCell(beta,1)
                  DispBeta(2) = DispY + SuperCell(beta,2)
                  DispBeta(3) = DispZ + SuperCell(beta,3)

                              distance =
     *              (SuperCell(alpha,1)-DispBeta(1))**2+
     *              (SuperCell(alpha,2)-DispBeta(2))**2+
     *              (SuperCell(alpha,3)-DispBeta(3))**2
                  distance = sqrt(distance)
                  if(TypeNN .eq. TypeA) then
                     if ((distance .gt. (SelfSNNDist+epsilon)).or.
     *                   (distance .lt. epsilon)) then
                        continue

                     elseif ((distance .gt. (SelfNNDist+epsilon))
     *                      .and. (distance .lt. (SelfSNNDist
     *                      - epsilon))) then
                        SelfSNNDist = distance
                        NSSNN(alpha) = 1
                        SList1(alpha,NSSNN(alpha)) = beta
                        SSNN(alpha,NSSNN(alpha),1) =
     *                       DispBeta(1) - SuperCell(alpha,1)
                        SSNN(alpha,NSSNN(alpha),2) =
     *                       DispBeta(2) - SuperCell(alpha,2)
                        SSNN(alpha,NSSNN(alpha),3) =
     *                       DispBeta(3) - SuperCell(alpha,3)

                     elseif ((distance .gt. (SelfSNNDist-epsilon))
     *                      .and. (distance .lt. (SelfSNNDist +
     *                      epsilon))) then
                        NSSNN(alpha) = NSSNN(alpha) + 1
                        SList1(alpha,NSSNN(alpha)) = beta
                        SSNN(alpha,NSSNN(alpha),1) =
     *                       DispBeta(1) - SuperCell(alpha,1)
                        SSNN(alpha,NSSNN(alpha),2) =
     *                       DispBeta(2) - SuperCell(alpha,2)
                        SSNN(alpha,NSSNN(alpha),3) =
     *                       DispBeta(3) - SuperCell(alpha,3)

                     elseif (distance .lt. (SelfNNDist-epsilon))
     *                      then
                        SelfNNDist = distance
                        NSNN(alpha) = 1
                        SList(alpha,NSNN(alpha)) = beta
                        SNN(alpha,NSNN(alpha),1) =
     *                       DispBeta(1) - SuperCell(alpha,1)
                        SNN(alpha,NSNN(alpha),2) =
     *                       DispBeta(2) - SuperCell(alpha,2)
                        SNN(alpha,NSNN(alpha),3) =
     *                       DispBeta(3) - SuperCell(alpha,3)

                     elseif ((distance .gt. (SelfNNDist-epsilon))
```

```
     *                          .and. (distance .lt.
     *                          (SelfNNDist+epsilon))) then


                              NSNN(alpha) = NSNN(alpha) + 1
                              SList(alpha,NSNN(alpha)) = beta
                              SNN(alpha,NSNN(alpha),1) =
     *                             DispBeta(1) - SuperCell(alpha,1)
                              SNN(alpha,NSNN(alpha),2) =
     *                             DispBeta(2) - SuperCell(alpha,2)
                              SNN(alpha,NSNN(alpha),3) =
     *                             DispBeta(3) - SuperCell(alpha,3)
                           else
                              print *,'error in self nearest neighbour'
                              stop
                           endif

                        else

C     so they are different

                           if (distance .gt. (DifDist+epsilon)) then
                              continue

                           elseif (distance .lt. (DifDist-epsilon)) then
                              DifDist = distance
                              NDNN(alpha) = 1
                              DList(alpha,NDNN(alpha)) = beta
                              DNN(alpha,NDNN(alpha),1) =
     *                             DispBeta(1) - SuperCell(alpha,1)
                              DNN(alpha,NDNN(alpha),2) =
     *                             DispBeta(2) - SuperCell(alpha,2)
                              DNN(alpha,NDNN(alpha),3) =
     *                             DispBeta(3) - SuperCell(alpha,3)

                           elseif ((distance .gt. (DifDist-epsilon)) .and.
     *                             (distance .lt. (DifDist+epsilon))) then
                              NDNN(alpha) = NDNN(alpha) + 1
                              DList(alpha,NDNN(alpha)) = beta
                              DNN(alpha,NDNN(alpha),1) =
     *                             DispBeta(1) - SuperCell(alpha,1)
                              DNN(alpha,NDNN(alpha),2) =
     *                             DispBeta(2) - SuperCell(alpha,2)
                              DNN(alpha,NDNN(alpha),3) =
     *                             DispBeta(3) - SuperCell(alpha,3)
                           else
                              print *,'error in dif. nearest neighbour'
                           endif
                        endif
  50                 continue
  40              continue
  30           continue
  20        continue
  10     continue
         return
         end


C        ---------------------------------------------------------------
C        this subroutine finds the appropriate force constant matrix
C        depenging on the type of the atom, the nearest neighbour and
C        the vector(x,y,z) between them

         subroutine AppMat(AM,type,typeNN,x,y,z)
         include "hfparamPert.com"
         include "hfcomPert.dec"
```

```
      include "hfcomPert.def"


      double precision AM(3,3)
      integer type,typeNN,x,y,z


C     initialize local varibles

      double precision ap,bp,cp,ad,bd,cd,g,h
      double precision gp,hp,gd,hd
      integer i,j

      double precision PPppxy(3,3),PPppxz(3,3),PPppyz(3,3)
      double precision PPpnxy(3,3),PPpnxz(3,3),PPpnyz(3,3)

      double precision DDppxy(3,3),DDppxz(3,3),DDppyz(3,3)
      double precision DDpnxy(3,3),DDpnxz(3,3),DDpnyz(3,3)

      double precision PDx(3,3),PDy(3,3),PDz(3,3)

      double precision SPPx(3,3),SPPy(3,3),SPPz(3,3)
      double precision SDDx(3,3),SDDy(3,3),SDDz(3,3)

      do 1 i=1,3
         do 2 j=1,3
            AM(i,j) = 0.0
 2       continue
 1    continue

C     give the rather cumbersome definitions of the force constant
C     matrices.


      ap = 12104.0
      bp = 18483.0
      cp = 1184.0

      ad = 269.0
      bd = 1633.0
      cd = -308.0

      g = 1697.0
      h = 2315.0

      gp = 4355.0
      hp = -1484.0

      gd = 1911.0
      hd = -164.0

      SPPx(1,1) = gp
      SPPx(1,2) = 0.0
      SPPx(1,3) = 0.0
      SPPx(2,1) = 0.0
      SPPx(2,2) = hp
      SPPx(2,3) = 0.0
      SPPx(3,1) = 0.0
      SPPx(3,2) = 0.0
      SPPx(3,3) = hp

      SPPy(1,1) = hp
      SPPy(1,2) = 0.0
      SPPy(1,3) = 0.0
      SPPy(2,1) = 0.0
      SPPy(2,2) = gp
```

```
SPPy(2,3) = 0.0
SPPy(3,1) = 0.0
SPPy(3,2) = 0.0
SPPy(3,3) = hp

SPPz(1,1) = hp
SPPz(1,2) = 0.0
SPPz(1,3) = 0.0
SPPz(2,1) = 0.0
SPPz(2,2) = hp
SPPz(2,3) = 0.0
SPPz(3,1) = 0.0
SPPz(3,2) = 0.0
SPPz(3,3) = gp


SDDx(1,1) = gd
SDDx(1,2) = 0.0
SDDx(1,3) = 0.0
SDDx(2,1) = 0.0
SDDx(2,2) = hd
SDDx(2,3) = 0.0
SDDx(3,1) = 0.0
SDDx(3,2) = 0.0
SDDx(3,3) = hd

SDDy(1,1) = hd
SDDy(1,2) = 0.0
SDDy(1,3) = 0.0
SDDy(2,1) = 0.0
SDDy(2,2) = gd
SDDy(2,3) = 0.0
SDDy(3,1) = 0.0
SDDy(3,2) = 0.0
SDDy(3,3) = hd

SDDz(1,1) = hd
SDDz(1,2) = 0.0
SDDz(1,3) = 0.0
SDDz(2,1) = 0.0
SDDz(2,2) = hd
SDDz(2,3) = 0.0
SDDz(3,1) = 0.0
SDDz(3,2) = 0.0
SDDz(3,3) = gd

PDx(1,1) = g
PDx(1,2) = 0
PDx(1,3) = 0
PDx(2,1) = 0
PDx(2,2) = h
PDx(2,3) = 0
PDx(3,1) = 0
PDx(3,2) = 0
PDx(3,3) = h

PDy(1,1) = h
PDy(1,2) = 0
PDy(1,3) = 0
PDy(2,1) = 0
PDy(2,2) = g
PDy(2,3) = 0
PDy(3,1) = 0
PDy(3,2) = 0
PDy(3,3) = h
```

```
PDz(1,1) = h
PDz(1,2) = 0
PDz(1,3) = 0
PDz(2,1) = 0
PDz(2,2) = h
PDz(2,3) = 0
PDz(3,1) = 0
PDz(3,2) = 0
PDz(3,3) = g

PPppxy(1,1) = ap
PPppxy(1,2) = bp
PPppxy(1,3) = 0
PPppxy(2,1) = bp
PPppxy(2,2) = ap
PPppxy(2,3) = 0
PPppxy(3,1) = 0
PPppxy(3,2) = 0
PPppxy(3,3) = cp

PPppxz(1,1) = ap
PPppxz(1,2) = 0
PPppxz(1,3) = bp
PPppxz(2,1) = 0
PPppxz(2,2) = cp
PPppxz(2,3) = 0
PPppxz(3,1) = bp
PPppxz(3,2) = 0
PPppxz(3,3) = ap

PPppyz(1,1) = cp
PPppyz(1,2) = 0
PPppyz(1,3) = 0
PPppyz(2,1) = 0
PPppyz(2,2) = ap
PPppyz(2,3) = bp
PPppyz(3,1) = 0
PPppyz(3,2) = bp
PPppyz(3,3) = ap

PPpnxy(1,1) = ap
PPpnxy(1,2) = -bp
PPpnxy(1,3) = 0
PPpnxy(2,1) = -bp
PPpnxy(2,2) = ap
PPpnxy(2,3) = 0
PPpnxy(3,1) = 0
PPpnxy(3,2) = 0
PPpnxy(3,3) = cp

PPpnxz(1,1) = ap
PPpnxz(1,2) = 0
PPpnxz(1,3) = -bp
PPpnxz(2,1) = 0
PPpnxz(2,2) = cp
PPpnxz(2,3) = 0
PPpnxz(3,1) = -bp
PPpnxz(3,2) = 0
PPpnxz(3,3) = ap

PPpnyz(1,1) = cp
PPpnyz(1,2) = 0
PPpnyz(1,3) = 0
PPpnyz(2,1) = 0
PPpnyz(2,2) = ap
PPpnyz(2,3) = -bp
```

```
PPpnyz(3,1) = 0
PPpnyz(3,2) = -bp
PPpnyz(3,3) = ap

DDppxy(1,1) = ad
DDppxy(1,2) = bd
DDppxy(1,3) = 0
DDppxy(2,1) = bd
DDppxy(2,2) = ad
DDppxy(2,3) = 0
DDppxy(3,1) = 0
DDppxy(3,2) = 0
DDppxy(3,3) = cd

DDppxz(1,1) = ad
DDppxz(1,2) = 0
DDppxz(1,3) = bd
DDppxz(2,1) = 0
DDppxz(2,2) = cd
DDppxz(2,3) = 0
DDppxz(3,1) = bd
DDppxz(3,2) = 0
DDppxz(3,3) = ad

DDppyz(1,1) = cd
DDppyz(1,2) = 0
DDppyz(1,3) = 0
DDppyz(2,1) = 0
DDppyz(2,2) = ad
DDppyz(2,3) = bd
DDppyz(3,1) = 0
DDppyz(3,2) = bd
DDppyz(3,3) = ad

DDpnxy(1,1) = ad
DDpnxy(1,2) = -bd
DDpnxy(1,3) = 0
DDpnxy(2,1) = -bd
DDpnxy(2,2) = ad
DDpnxy(2,3) = 0
DDpnxy(3,1) = 0
DDpnxy(3,2) = 0
DDpnxy(3,3) = cd


DDpnxz(1,1) = ad
DDpnxz(1,2) = 0
DDpnxz(1,3) = -bd
DDpnxz(2,1) = 0
DDpnxz(2,2) = cd
DDpnxz(2,3) = 0
DDpnxz(3,1) = -bd
DDpnxz(3,2) = 0
DDpnxz(3,3) = ad

DDpnyz(1,1) = cd
DDpnyz(1,2) = 0
DDpnyz(1,3) = 0
DDpnyz(2,1) = 0
DDpnyz(2,2) = ad
DDpnyz(2,3) = -bd
DDpnyz(3,1) = 0
DDpnyz(3,2) = -bd
DDpnyz(3,3) = ad
```

```
     if(type.eq.46 .and. typeNN .eq.46) then
        if( (x.eq.1 .and. y.eq.1 .and. z.eq.0) .or.
    *        (x.eq.-1 .and. y.eq.-1 .and. z.eq.0) ) then
           do 10 i=1,3
              do 20 j=1,3
                 AM(i,j) = PPppxy(i,j)
20            continue
10         continue

        elseif( (x.eq.1 .and. y.eq.0 .and. z.eq.1) .or.
    *           (x.eq.-1 .and. y.eq.0 .and. z.eq.-1)) then
           do 30 i=1,3
              do 40 j=1,3
                 AM(i,j) = PPppxz(i,j)
40            continue
30         continue

        elseif( (x.eq.0 .and. y.eq.1 .and. z.eq.1) .or.
    *           (x.eq.0 .and. y.eq.-1 .and. z.eq.-1)) then
           do 50 i=1,3
              do 60 j=1,3
                 AM(i,j) = PPppyz(i,j)
60            continue
50         continue

        elseif( (x.eq.1 .and. y.eq.-1 .and. z.eq.0) .or.
    *           (x.eq.-1 .and. y.eq.1 .and. z.eq.0) ) then
           do 70 i=1,3
              do 80 j=1,3
                 AM(i,j) = PPpnxy(i,j)
80            continue
70         continue

        elseif( (x.eq.1 .and. y.eq.0 .and. z.eq.-1) .or.
    *           (x.eq.-1 .and. y.eq.0 .and. z.eq.1)) then
           do 90 i=1,3
              do 100 j=1,3
                 AM(i,j) = PPpnxz(i,j)
100           continue
90         continue

        elseif( (x.eq.0 .and. y.eq.1 .and. z.eq.-1) .or.
    *           (x.eq.0 .and. y.eq.-1 .and. z.eq.1)) then
           do 110 i=1,3
              do 120 j=1,3
                 AM(i,j) = PPpnyz(i,j)
120           continue
110        continue

        elseif( (x.eq.2 .or. x.eq.-2) .and. y.eq.0 .and.
    *           z.eq.0) then
           do 122 i=1,3
              do 123 j=1,3
                 AM(i,j) = SPPx(i,j)
123           continue
122        continue

        elseif( x.eq.0 .and. (y.eq.2 .or. y.eq.-2) .and.
    *           z.eq.0) then
           do 124 i=1,3
              do 125 j=1,3
                 AM(i,j) = SPPy(i,j)
125           continue
124        continue

        elseif( x.eq.0 .and. y.eq.0 .and. (z.eq.2 .or.
```

```
     *           z.eq.-2)) then
                do 126 i=1,3
                   do 127 j=1,3
                      AM(i,j) = SPPz(i,j)
 127              continue
 126           continue
           else
C             print *,'error in 46-46 interaction'
C             print *,x,y,z
C     stop
           endif

          elseif( (type.eq.46 .and. typeNN.eq.1) .or.
     *         (type.eq.1 .and. typeNN.eq.46)) then

           if((x.eq.1 .or. x.eq.-1) .and. y.eq.0 .and. z.eq.0) then
              do 130 i=1,3
                 do 140 j=1,3
                    AM(i,j) = PDx(i,j)
 140             continue
 130          continue

           elseif(x.eq.0 .and. (y.eq.1 .or. y.eq.-1) .and. z.eq.0) then
              do 150 i=1,3
                 do 160 j=1,3
                    AM(i,j) = PDy(i,j)
 160             continue
 150          continue

           elseif(x.eq.0 .and. y.eq.0 .and. (z.eq.1 .or. z.eq.-1)) then
              do 170 i=1,3
                 do 180 j=1,3
                    AM(i,j) = PDz(i,j)
 180             continue
 170          continue


           else
C             print *,'error in 46-1 interaction'
C             print *,x,y,z
C     stop
           endif


          elseif (type.eq.1 .and. typeNN.eq.1) then
              if( (x.eq.1 .and. y.eq.1 .and. z.eq.0) .or.
     *         (x.eq.-1 .and. y.eq.-1 .and. z.eq.0) ) then
                do 250 i=1,3
                   do 260 j=1,3
                      AM(i,j) = DDppxy(i,j)
 260              continue
 250           continue

           elseif( (x.eq.1 .and. y.eq.0 .and. z.eq.1) .or.
     *             (x.eq.-1 .and. y.eq.0 .and. z.eq.-1) ) then
                do 270 i=1,3
                   do 280 j=1,3
                      AM(i,j) = DDppxz(i,j)
 280              continue
 270           continue

           elseif( (x.eq.0 .and. y.eq.1 .and. z.eq.1) .or.
     *             (x.eq.0 .and. y.eq.-1 .and. z.eq.-1)) then
                do 290 i=1,3
                   do 300 j=1,3
                      AM(i,j) = DDppyz(i,j)
```

```
 300          continue
 290       continue

           elseif( (x.eq.1 .and. y.eq.-1 .and. z.eq.0) .or.
     *             (x.eq.-1 .and. y.eq.1 .and. z.eq.0) ) then
              do 310 i=1,3
                 do 320 j=1,3
                    AM(i,j) = DDpnxy(i,j)
 320             continue
 310          continue

           elseif( (x.eq.1 .and. y.eq.0 .and. z.eq.-1) .or.
     *             (x.eq.-1 .and. y.eq.0 .and. z.eq.1)) then
              do 330 i=1,3
                 do 340 j=1,3
                    AM(i,j) = DDpnxz(i,j)
 340             continue
 330          continue

           elseif( (x.eq.0 .and. y.eq.1 .and. z.eq.-1) .or.
     *             (x.eq.0 .and. y.eq.-1 .and. z.eq.1)) then
              do 350 i=1,3
                 do 360 j=1,3
                    AM(i,j) = DDpnyz(i,j)
 360             continue
 350          continue

           elseif( (x.eq.2 .or. x.eq.-2) .and. y.eq.0 .and.
     *             z.eq.0) then
              do 370 i=1,3
                 do 380 j=1,3
                    AM(i,j) = SDDx(i,j)
 380             continue
 370          continue

           elseif( x.eq.0 .and. (y.eq.2 .or. y.eq.-2) .and.
     *             z.eq.0) then

              do 390 i=1,3
                 do 400 j=1,3
                    AM(i,j) = SDDy(i,j)
 400             continue
 390          continue

           elseif( x.eq.0 .and. y.eq.0 .and. (z.eq.2 .or.
     *             z.eq.-2)) then
              do 410 i=1,3
                 do 420 j=1,3
                    AM(i,j) = SDDz(i,j)
 420             continue
 410          continue
           else
C               print *,'error in 1-1 interaction'
C               print *,x,y,z
C     stop
           endif

         else
            print *,'major error in call AppMat'
         endif
         return
         end

C     ---------------------------------------------------------------
C     this subroutine loads the data
```

```
      subroutine LoadData()

      include "hfparamPert.com"
      include "hfcomPert.dec"
      include "hfcomPert.def"

C     declare local variables
      integer i
      integer x,y,z,dum1,dum2,type

      open(7,file = 'PdD.cell')

      do 10 i=1, NASC

         read (7,*) x,y,z,dum1,type,dum2
C     The cell data comes in with a lattice constant of 20. we need
C     to divide by 10 to get it to a lattice constant of 2 and 1 is
C     added because the origin is at (0,0,0), but in an array labelled
C     by (1,1,1)

         SuperCell(i,1) = x/10
         SuperCell(i,2) = y/10
         SuperCell(i,3) = z/10

         TypeSC(i) = -type

 10   continue
      close(7)
      return
      end
```

```
C ***********************************************************************
C ************************** Experimental Data ***************************
C ***********************************************************************

C Lattice Constant: (Angstroms)
D  3.89
D 0.

C Elastic Constants: (10^12 dyn/cm^2)
C C11 C12 C44
D 2.341 1.761 .712
D 0. 0. 0.

C Sublimation Energy: (eV)
D 3.91
D 0.

C Vacancy Formation Energy: (eV)
D 1.4
D 0.

C ***********************************************************************
C ************************** Embedding Function **************************
C ***********************************************************************
C Analytical Expression for Embedding Energy: (.TRUE or .FALSE)
D 0

C Spline Fit Knots:
C Number of Spline Knots
D 5

C Equlibrium Density: (cm-3)
D 0.01518e+24

C i Rho  F(rho)
D 1  0.0      0.0
D 2  0.5     -3.117
D 3  1.0     -4.697
D 4  2.0     -3.015
D 5  2.3      0.0

C ***********************************************************************
C ************************** Pair Potential ** ***************************
C ***********************************************************************
C Analytical Expression for Pair Potential: (.TRUE or .FALSE)
C D 0

C Spline Fit Knots:
C Number of Spline Knots
D 5

C i r Z(r)
D 1  0.0     46.0
D 2  0.430    6.6580
D 3  0.650    0.2980
D 4  0.710    0.1530
D 5  0.85     0.0

C ***********************************************************************
C **************Parameters used to calculate density*****************
C ***********************************************************************

C # of shells involved in the density
D 2

C contribution from each shell
```

```
D .65 9.35


C ..................................
C Wave functions from Clementi Tables
C Units:   zsi is in inverse angstroms (NOTE: Clementi uses atomic units)
C ..................................

D Units of Zsi (in inverse centimeters)= 1.e+8

D N= 5     L= 0  Norb= 10
C i   N(i)      zsi(i)      C(i)
C -   ----      ------      ----
D 1   1         89.21928    -0.00071
D 2   1         61.90983     0.02424
D 3   2         40.12741     0.16808
D 4   2         38.42703    -0.24234
D 5   3         26.92741    -0.01686
D 6   3         18.39798     0.19178
D 7   4         10.68346    -0.27759
D 8   4          7.24112    -0.02257
D 9   5          4.20229     0.55209
D 10  5          2.33989     0.57052

D N= 4     L= 2  Norb= 4
C i   N(i)      zsi(i)      C(i)
C -   ----      ------      ----
D 1   3         29.86560    -0.08721
D 2   3         16.80195    -0.23876
D 3   4          9.02038     0.57074
D 4   4          4.67147     0.58201
```

```
      subroutine dfunc(DynX,df)

      include 'eamparam.com'
      include 'hfparamPert.com'
      include 'eamcom.def'
      include 'hfcomPert.def'
      include 'eamcom.dec'
      include 'hfcomPert.dec'

      double precision df(3*NDynX)
      double precision DynX(NDynX*3)

      double precision dTotPhidr(3*NDynX),dFdr(3*NDynX)
      integer i
      external dCalcPhi, dCalcF
C     convert to supercell format

C      print *,'IN DFUNC'
      do 10 i=1,NDynX
         SuperCell(i,1) = DynX((i-1)*3 + 1)
         SuperCell(i,2) = DynX((i-1)*3 + 2)
         SuperCell(i,3) = DynX((i-1)*3 + 3)
 10   continue

      call NNeighbour()


C      print *,'calling CalcDphi'
      call CalcDphi(dTotPhidr)
C      print *,'called CalcDphi'
C      print *,'calling CalcDf'
      call CalcDf(dFdr)
C      print *,'called CalcDf'
C      print *,'DF'
      do 50 i=1,3*NDynX
         df(i) = (dTotPhidr(i) + dFdr(i))*1.0d10
C         print *,dTotPhidr(i),dFdr(i),df(i)
 50   continue

C      print *,'LEAVING DFUNC'
      return
      end


C     ----------------------------------------------------------------
      subroutine CalcDphi(dTotPhidr)

      include 'eamparam.com'
      include 'hfparamPert.com'
      include 'eamcom.def'
      include 'hfcomPert.def'
      include 'eamcom.dec'
      include 'hfcomPert.dec'

      double precision dTotPhidr(3*NDynX)

      double precision r,dPhidr
      double precision z1,dz1dr,d2z1dr2
      double precision z2,dz2dr,d2z2dr2
      integer Type1,Type2
      integer i,j

C      print *,'IN CALCDPHI'
      do 5 i=1,3*NDynX
         dTotPhidr(i) = 0.0
 5    continue
```

```
C     calculate the pair potential by going only to the nearest
C     neighbours

      do 10 i=1,NDynX
         Type1 = TypeSC(i)
         do 30 j=1,NSNN(i)
            Type2 = TypeSC(SList(i,j))
C            print *,'Type1,Type2,i,j'
C            print *,Type1,Type2,i,j
            r = sqrt(SNN(i,j,1)**2 + SNN(i,j,2)**2 + SNN(i,j,3)**2)
            if (Type1.eq.1 .and. Type2.eq.1) then
               call ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               dz1dr = dz1dr*e/(ang)
               dPhidr =  (2*r*ang*z1*dz1dr - z1*z1)/(r*ang*r*ang)
            elseif ((Type1.eq.1 .and. Type2.eq.46) .or. (Type1.eq.46
     *              .and.Type2.eq.1)) then
               call ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               dz1dr = dz1dr*e/(ang)
               call ZH(r*ang,z2,dz2dr,d2z2dr2)
               z2 = z2*e
               dz2dr = dz2dr*e/(ang*a0)
               dPhidr = (r*ang*z1*dz2dr + r*ang*z2*dz1dr - z1*z2)
     *              /(r*ang*r*ang)
            elseif (Type1.eq.46 .and. Type2.eq.46) then
               call ZH(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               dz1dr = dz1dr*e/(ang*a0)
               dPhidr = (2*r*ang*z1*dz1dr - z1*z1)/(r*ang*r*ang)
            else
               print *,'Type1,Type2'
               print *,Type1,Type2
               print *,'error in CalcDPhi1'
               stop
            endif

            dTotPhidr((i-1)*3+1)   = dTotPhidr((i-1)*3+1)
     *           + dPhidr*SNN(i,j,1)/r
            dTotPhidr((i-1)*3+2) = dTotPhidr((i-1)*3+2)
     *           + dPhidr*SNN(i,j,2)/r
            dTotPhidr((i-1)*3+3) = dTotPhidr((i-1)*3+3)
     *           + dPhidr*SNN(i,j,3)/r
 30      continue

         do 40 j=1,NDNN(i)
            Type2 = TypeSC(DList(i,j))
            r = sqrt(DNN(i,j,1)**2 + DNN(i,j,2)**2 + DNN(i,j,3)**2)
            if (Type1.eq.1 .and. Type2.eq.1) then
               call ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               dz1dr = dz1dr*e/ang
               dPhidr =  (2*r*ang*z1*dz1dr - z1*z1)/(r*ang*r*ang)
            elseif ((Type1.eq.1 .and. Type2.eq.46) .or. (Type1.eq.46
     *              .and. Type2.eq.1)) then
               call ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               dz1dr = dz1dr*e/(ang)
               call ZH(r*ang,z2,dz2dr,d2z2dr2)
               z2 = z2*e
               dz2dr = dz2dr*e/(ang*a0)
               dPhidr = (r*ang*z1*dz2dr + r*ang*z2*dz1dr - z1*z2)
     *              /(r*ang*r*ang)
            elseif (Type1.eq.46 .and. Type2.eq.46) then
               call ZH(r*ang,z1,dz1dr,d2z1dr2)
```

```
               z2 = z2*e
               dz2dr = dz2dr*e/(ang*a0)
               dPhidr = (2*r*ang*z1*dz1dr - z1*z1)/(r*ang*r*ang)
            else
               print *,'Type1,Type2'
               print *,Type1,Type2
               print *,'error in CalcDPhi2'
               stop
            endif

            dTotPhidr((i-1)*3+1)   = dTotPhidr((i-1)*3+1)
     *           + dPhidr*DNN(i,j,1)/r
            dTotPhidr((i-1)*3+2) = dTotPhidr((i-1)*3+2)
     *           + dPhidr*DNN(i,j,2)/r
            dTotPhidr((i-1)*3+3) = dTotPhidr((i-1)*3+3)
     *           + dPhidr*DNN(i,j,3)/r

 40      continue
 10    continue

C       print *,'LEAVING CALCDPHI'
      return
      end

C     ----------------------------------------------------------------

      subroutine CalcDf(dFdx)
      include 'eamparam.com'
      include 'hfparamPert.com'
      include 'eamcom.def'
      include 'hfcomPert.def'
      include 'eamcom.dec'
      include 'hfcomPert.dec'

      double precision dFdx(3*NDynX)

      double precision TotalRho,TotalRhoK
      double precision r,rho,drhodr,d2rhodr2
      double precision r,rhoK,drhoKdr,d2rhoKdr2
      double precision F,dFdrho,d2Fdrho2
      double precision FK,dFKdrho,d2FKdrho2
      double precision xkmxi,ykmyi,zkmzi

      integer TypeI,TypeJ,TypeK
      integer i,j,k

C       print *,'IN CALCDF'
      do 10 k=1,NDynX
         TotalRhoK = 0.0

         TypeK = TypeSC(k)
         do 30 j=1,NSNN(k)
            TypeJ = TypeSC(SList(k,j))
            r = sqrt(SNN(k,j,1)**2 + SNN(k,j,2)**2 + SNN(k,j,3)**2)
            if (TypeJ.eq.1) then
               call calcrhoI(r,rhoK,drhoKdr,d2rhoKdr2)
            elseif(TypeJ.eq.46) then
               call calcrho(r*ang,rhoK,drhoKdr,d2rhoKdr2)
C               print *,'r,rhoK,drhoKdr'
C               print *,r,rhoK,drhoKdr
            else
               print *,TypeJ
               print *,'error1'
               stop
            endif
```

```
            TotalRhoK = TotalRhoK + rhoK
 30       continue

          do 40 j=1,NSSNN(k)
             TypeJ = TypeSC(SList1(k,j))
             r = sqrt(SSNN(k,j,1)**2 + SSNN(k,j,2)**2 + SSNN(k,j,3)**2)
             if (TypeJ.eq.1) then
                call calcrhoI(r,rhoK,drhoKdr,d2rhoKdr2)
             elseif(TypeJ.eq.46) then
                call calcrho(r*ang,rhoK,drhoKdr,d2rhoKdr2)
C                print *,'r,rhoK,drhoKdr'
C                print *,r,rhoK,drhoKdr
             else
                print *,TypeJ
                print *,'error2'
                stop
             endif

             TotalRhoK = TotalRhoK + rhoK
 40       continue

          do 45 j=1,NSTNN(k)
             TypeJ = TypeSC(SList2(k,j))
             r = sqrt(STNN(k,j,1)**2+STNN(k,j,2)**2+STNN(k,j,3)**2)
             if (TypeJ.eq.1) then
                call calcrhoI(r,rhoK,drhoKdr,d2rhoKdr2)
             elseif(TypeJ.eq.46) then
                call calcrho(r*ang,rhoK,drhoKdr,d2rhoKdr2)
C                print *,'r,rhoK,drhoKdr'
C                print *,r,rhoK,drhoKdr
             else
                print *,TypeJ
                print *,'error1'
                stop
             endif

             TotalRhoK = TotalRhoK + rhoK
 45       continue

          do 50 j=1,NDNN(k)
             TypeJ = TypeSC(DList(k,j))
             r = sqrt(DNN(k,j,1)**2 + DNN(k,j,2)**2 + DNN(k,j,3)**2)
             if (TypeJ.eq.1) then
                call calcrhoI(r,rhoK,drhoKdr,d2rhoKdr2)
             elseif(TypeJ.eq.46) then
                call calcrho(r*ang,rhoK,drhoKdr,d2rhoKdr2)
             else
                print *,TypeJ
                print *,'error3'
                stop
             endif

             TotalRhoK = TotalRhoK + rhoK
 50       continue


          if (TypeK.eq.1) then
             call FI(TotalRhoK*densunit*densunit*densunit,FK,dFKdrho,
     *            d2FKdrho2)
             FK = FK*eV
             dFKdrho = dFKdrho*eV/(densunit*densunit*densunit)
          elseif(TypeK.eq.46) then
             call FH(TotalRhoK*densunit*densunit*densunit,FK,dFKdrho,
     *            d2FKdrho2)
C             print *,'TotalRhoK,FK,dFKdrho'
C             print *,TotalRhoK,FK,dFKdrho
```

```
            FK = FK*eV
            dFKdrho = dFKdrho*eV/(rhobarH)
C            print *,TotalRhoK*densunit*densunit*densunit,FK,dFKdrho
         else
            print *,TypeK
            print *,'error4'
            stop
         endif

         do 60 i=1,NASC
            if (i.eq.k) then
               continue
            else
               TotalRho = 0.0
               TypeI = TypeSC(i)
               do 70 j=1,NSNN(i)
                  TypeJ = TypeSC(SList(i,j))
                  r = sqrt(SNN(i,j,1)**2+SNN(i,j,2)**2+SNN(i,j,3)**2)
                  if (TypeJ.eq.1) then
                     call calcrhoI(r,rho,drhodr,d2rhodr2)
                  elseif(TypeJ.eq.46) then
                     call calcrho(r*ang,rho,drhodr,d2rhodr2)
C                     print *,'r,rho,drhodr'
C                     print *,r,rho,drhodr
                  else
                     print *,TypeJ
                     print *,'error5'
                     stop
                  endif

                  TotalRho = TotalRho + rho
 70               continue

               do 80 j=1,NSSNN(i)
                  TypeJ = TypeSC(SList1(i,j))
                  r=sqrt(SSNN(i,j,1)**2+SSNN(i,j,2)**2+SSNN(i,j,3)**2)
                  if (TypeJ.eq.1) then
                     call calcrhoI(r,rho,drhodr,d2rhodr2)
                  elseif(TypeJ.eq.46) then
                     call calcrho(r*ang,rho,drhodr,d2rhodr2)
C                     print *,'r,rho,drhodr'
C                     print *,r,rho,drhodr
                  else
                     print *,TypeJ
                     print *,'error6'
                     stop
                  endif

                  TotalRho = TotalRho + rho
 80               continue

               do 85 j=1,NSTNN(i)
                  TypeJ = TypeSC(SList2(i,j))
                  r=sqrt(STNN(i,j,1)**2+STNN(i,j,2)**2+STNN(i,j,3)**2)
                  if (TypeJ.eq.1) then
                     call calcrhoI(r,rho,drhodr,d2rhodr2)
                  elseif(TypeJ.eq.46) then
                     call calcrho(r*ang,rho,drhodr,d2rhodr2)
C                     print *,'r,rho,drhodr'
C                     print *,r,rho,drhodr
                  else
                     print *,TypeJ
                     print *,'error5'
                     stop
                  endif
```

```
                  TotalRho = TotalRho + rho
 85               continue

                  do 90 j=1,NDNN(i)
                     TypeJ = TypeSC(DList(i,j))
                     r = sqrt(DNN(i,j,1)**2+DNN(i,j,2)**2+DNN(i,j,3)**2)
                     if (TypeJ.eq.1) then
                        call calcrhoI(r,rho,drhodr,d2rhodr2)
                     elseif(TypeJ.eq.46) then
                        call calcrho(r*ang,rho,drhodr,d2rhodr2)
                     else
                        print *,TypeJ
                        print *,'error7'
                        stop
                     endif

                     TotalRho = TotalRho + rho
 90               continue


                  if (TypeI.eq.1) then
                     call FI(TotalRho*densunit*densunit*densunit,F,dFdrho,
     *                    d2Fdrho2)
                     F = F*eV
                     dFdrho = F*eV/(densunit*densunit*densunit)
                  elseif(TypeI.eq.46) then
                     call FH(TotalRho*densunit*densunit*densunit,F,dFdrho,
     *                    d2Fdrho2)
C                     print *,'TotalRho,F,dFdrho'
C                     print *,TotalRho,F,dFdrho
                     F = F*eV
                     dFdrho = dFdrho*eV/rhobarH
C                     print *,TotalRho*densunit*densunit*densunit,F,dFdrho
                  else
                     print *,TypeI
                     print *,'error8'
                     stop
                  endif

                  xkmxi = SuperCell(k,1) - SuperCell(i,1)
                  ykmyi = SuperCell(k,2) - SuperCell(i,2)
                  zkmzi = SuperCell(k,3) - SuperCell(i,3)

                  r = sqrt(xkmxi**2 + ykmyi**2 + zkmzi**2)
C                  if(TypeK.eq.TypeI) then
C                     if (r.gt.(1.1*a0)) then
C                        xkmxi = 0.0
C                        ykmyi = 0.0
C                        zkmzi = 0.0
C                     endif
C                  else
C                     if(r.gt.(0.7*a0)) then
C                        xkmxi = 0.0
C                        ykmyi = 0.0
C                        zkmzi = 0.0
C                     endif
C                  endif

                  if (TypeK.eq.1) then
                     call calcrhoI(r,rhoK,drhoKdr,d2rhoKdr2)
                     drhoKdr = drhoKdr*(densunit)**4
                  elseif(TypeK.eq.46) then
                     call calcrho(r*ang,rhoK,drhoKdr,d2rhoKdr2)
                     drhoKdr = drhoKdr*(densunit)**4
C                     print *,'drhoKdr'
C                     print *,drhoKdr
```

```
                else
                   print *,TypeK
                   print *,'error9'
                   stop
                endif

                if (TypeI.eq.1) then
                   call calcrhoI(r,rho,drhodr,d2rhodr2)
                   drhodr = drhodr*(densunit)**4
                elseif(TypeI.eq.46) then
                   call calcrho(r*ang,rho,drhodr,d2rhodr2)
                   drhodr = drhodr*(densunit)**4
C                   print *,'rhodr'
C                   print *,drhodr
                else
                   print *,TypeI
                   print *,'error10'
                   stop
                endif


C               print *,'should sum to zero'
C               print *,'K','I'
C               print *,k,i
C               print *,TypeI,TypeK
C               print *,dFdrho*drhodr*xkmxi/r
C               print *,dFKdrho*drhoKdr*xkmxi/r


                dFdx((k-1)*3 + 1) = dFdx((k-1)*3 + 1) +
     *               dFKdrho * drhoKdr * xkmxi/r +
     *               dFdrho * drhodr * xkmxi/r
C               print *,'dFdx',dFdx((k-1)*3+1)
C               print *,'xkmxi,ykmyi,zkmzi'
C               print *,xkmxi,ykmyi,zkmzi

C               print*,' '
                dFdx((k-1)*3 + 2) = dFdx((k-1)*3 + 2) +
     *               dFKdrho * drhoKdr * ykmyi/r +
     *               dFdrho * drhodr *ykmyi/r

                dFdx((k-1)*3 + 3) = dFdx((k-1)*3 + 3) +
     *               dFKdrho * drhoKdr * zkmzi/r +
     *               dFdrho * drhodr *zkmzi/r
C               print *,'dFdx,dFdy,dFdz'
C               print *,dFdx((k-1)*3+1),dFdx((k-1)*3+2),dFdx((k-1)*3+3)
             endif
 60       continue
 10    continue
C       print *,'LEAVING CALCDF'
       return
       end
```

```
C This is the main routine that calls various EAM functions in EAMFUNC.F
C Amongst the things that this calculates:
C
C  1. Calculates pair interaction formula given various thermodynamic
C     and material properties.
C
C  2. Lattice configurations as a function of stoichiometry of host
C     and impurity concentration.
C
C  3. Calculates force constants

      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C..declare local variables..
      integer ichoice



C...Load Data...
      call loaddata


C...Initialize relevant variables...
      call init


      call disp1

 1    continue
      print *,' '
      print *,'Options: '
      print *,' '
      print *,'  1.  Show Fit Parameters'
      print *,'  2.  Display  Parameters for F(rho) and Z(r)'
      print *,'  3.  Display Density Parameters'
      print *,'  4.  Output Embedding Function'
      print *,'  5.  Output Pair Potential'
      print *,'  6.  Output Density'
      print *,'  7.  Run Embedded Atom Model'
      print *,'  8.  Quit'

      print *,' '
      print *,'Input Choice'
      read(*,*,err=1)ichoice
      if ((ichoice .lt. 1) .or. (ichoice .gt. 8)) goto 1

      if (ichoice .eq. 1) then
         call disp1
      else if (ichoice .eq. 2) then
         call disp2
      else if (ichoice .eq. 3) then
         call disp3
      else if (ichoice .eq. 4) then
         call outputf
      else if (ichoice .eq. 5) then
         call outputz
      else if (ichoice .eq. 6) then
         call outputn
      else if (ichoice .eq. 7) then
C         call runmod
         call main3
      else
         stop
      endif
```

```
      goto 1
      end

C-----------------------------------------------------SUBROUTINE INIT------
      subroutine init
      external fact
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C..define local variables..
      integer fact
      integer i,j

C...Define Constants.....
C mp: g
C me: g
C e: esu
C eV: ergs
C MeV: 10^6 eV
C KeV: 10^6 eV
C hbar: erg-sec
C ang: cm
C barns: cm
C vth: cm/sec (electron)
C kT: erg
      mp=1.6726d-24
      mn=1.6726d-24
      me=9.1095d-28
      e=4.80324d-10
      eV=1.60219d-12
      MeV=eV*1.d+6
      KeV=eV*1.d+3
      hbar=1.05459d-27
      ang=1.d-8
      barns=1.d-24
      vth=4.72d+6
      kb=1.38d-16
      pimath=dacos(-1.d0)

C..initialize appropriate variables...
      calcsplF=.false.
      calcsplZ=.false.
      calcsplI=.false.

C Define Ceff(i,j) from C(i,j) so that factorials have to be evaluated
C only once

      do 20 i=1,Nshell
         do 10 j=1,Norb(i)
            Ceff(i,j)=C(i,j)*
     &      (2.d0*zsi(i,j))**(dfloat(nconfig(i,j))+.5d0) /
     &       dsqrt(dfloat(fact(2*nconfig(i,j))))
            print *,i,j,nconfig(i,j),Ceff(i,j)

 10      continue
 20   continue

      return
      end

C-----------------------------------------------------FUNCTION FACT -------
      integer function fact(n)

      integer i,n1,n
```

```
      if (n .lt. 0) then
         print *,'ERROR: FACT'
         stop
      endif

      n1=1
      do 10 i=1,n
         n1=n1*i
10    continue

      fact=n1
      return
      end
```

```
      double precision a0,c11,c12,c44,Es,Evacancy
      double precision a0fit,c11fit,c12fit,c44fit,Esfit,Evfit
      double precision mp,mn,me,e,eV,MeV,KeV,hbar,ang,barns,vth,kb,
     &                 pimath
      double precision rhofH,frhoH,d2FH
      double precision rhofI,frhoI,d2FI
      double precision rzH,zrH,d2ZH,rhobarH
      double precision rhobarI
      double precision B1H,B2H,B3H,B4H,B1I,B2I,B3I,B4I
      double precision pH,pI
      double precision densunit,zsi,C,Ceff,Ns
      double precision statlattice,HyAtom

      integer nspFH,nspZH
      integer nspFI,nspZI
      integer Nshell,elevel,angmom,Norb,nconfig
      logical calcsplF,calcsplZ,analyH
      logical calcsplI
      character*10 namhost,namimp
```

```
      common /exp/a0,c11,c12,c44,Es,Evacancy
      common /rel/statlattice(MT,3),HyAtom(3)
      common /fit/a0fit,c11fit,c12fit,c44fit,Esfit,Evfit
      common /constants/mp,mn,me,e,eV,MeV,KeV,hbar,ang,barns,vth,kb,
     &                  pimath
      common /names/namhost,namimp
      common /Fspline/rhofH(MAXKNOTS),frhoH(MAXKNOTS),d2FH(MAXKNOTS),
     &  calcsplF,nspFH, rhobarH,
     &  rhofI(MAXKNOTS),frhoI(MAXKNOTS),d2FI(MAXKNOTS),
     &  calcsplI,nspFI, rhobarI
      common /Fanaly/B1H,B2H,B3H,B4H,B1I,B2I,B3I,B4I,analyH

      common /Zspline/rzH(MAXKNOTS),zrH(MAXKNOTS),d2ZH(MAXKNOTS),
     &  calcsplZ,nspZH,nspZI
      common /Zanly/pH,pI

      common /density/densunit,Ns(10),zsi(10,15),C(10,15),Ceff(10,15),
     &  nconfig(10,15),Nshell,elevel(10),angmom(10),Norb(10)
```

```
C-----------------------------------------------SUBROUTINE FH ---------------
      subroutine FH(rho,F,dFdrho,d2Fdrho2)
C
C Calculates embedding functions using either the analytic fit or the
C spline fit parameters depending upon what is available
C
C rho is in angs^-3
C dFdrho is in Es*eV/angs^-3

      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C...Declaration of Arguments
      double precision rho, F, dFdrho,d2Fdrho2

C..declaration of local variables...
      double precision u

      if (analyH) then
         u=rho*ang*ang*ang
         call fanalH(u,F,dFdrho,d2Fdrho2)
      else
         u=rho/rhobarH
         call fsplineH(u,F,dFdrho,d2Fdrho2)
      endif

      return
      end


C-----------------------------------------------SUBROUTINE ZH ---------------
      subroutine ZH(r,Z,dZdr,d2Zdr2)
C
C Calculates pair potential using either the analytic fit or the
C spline fit parameters depending upon what is available
C
C r is given in terms of ang*a0.
C Z is in au. dZdr is in au/(ang*a0)



      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of Arguments
      double precision r,Z,dZdr,d2Zdr2

C..declaration of local variables...
      double precision u

      if (analyH) then
         u=r/ang
         call zanalH(u,Z,dZdr,d2Zdr2)
      else
         u=r/(ang*a0)
         call zsplineH(u,Z,dZdr,d2Zdr2)
      endif

      return
      end


C-----------------------------------------------SUBROUTINE FI ---------------
      subroutine FI(rho,F,dFdrho,d2Fdrho2)
```

```
C
C Calculates embedding functions using either the analytic fit or the
C spline fit parameters depending upon what is available
C
C rho is in angs^-3
C F is in eV.dFdrho is in eV/angs^-3

      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of Arguments
      double precision rho, F, dFdrho,d2Fdrho2

C..declaration of local variables...
      double precision u

C      if (analyI) then
         u=rho*ang*ang*ang
         call fanalI(u,F,dFdrho,d2Fdrho2)
C      else
C         call fsplineI(rho,F,dGdrho,d2Gdrho2)
C         print *,'SUBROUTINE FI: Feature not yet available'
C         stop
C      endif

      return
      end


C-----------------------------------------------SUBROUTINE ZI ---------------
      subroutine ZI(r,Z,dZdr,d2Zdr2)
C
C Calculates pair potential using either the analytic fit or the
C spline fit parameters depending upon what is available
C
C r is given in cm
C Z is given in au. dZdr in au/angs



      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of Arguments
      double precision r,Z,dZdr,d2Zdr2

C..declaration of local variables...
      double precision u

C      if (analyI) then
         u=r/ang
         call zanalI(u,Z,dZdr,d2Zdr2)
C      else
C         call zsplineI(r,Z,dZdr,d2Zdr2)
C         print *,'SUBROUTINE ZI: Feature not yet available'
C         stop
C      endif

      return
      end

C-----------------------------------------------SUBROUTINE CALCRHO--------
      subroutine calcrho(r,rho,drhodr,d2rhodr2)
C
```

```
C Calculates the local electron density using Hartree-Fock wave functions
C from Clementi
C
C r is in cm
C rho is in angs^-3.drhodr is in angs^-4

      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C..declaration of arguments...
      double precision r,rho,drhodr,d2rhodr2

C..declaration of local variables..
      double precision u
      double precision Rij,dRij,d2Rij,tmp1
      double precision rhoi,drhoi,d2rhoi
      integer i,j

      u=r*densunit
      rho=0.d0
      drhodr=0.d0
      d2rhodr2=0.d0
      do 10 i=1,Nshell
         rhoi=0.d0
         drhoi=0.d0
         d2rhoi=0.d0
         do 20 j=1,Norb(i)
            Rij=Ceff(i,j) * dexp(-zsi(i,j)*u)
            Rij = Rij*(u**(nconfig(i,j)-1))
            if (nconfig(i,j) .gt. 1) then
               tmp1=Ceff(i,j)*(u**(nconfig(i,j)-2))*dexp(-zsi(i,j)*u)
               dRij = tmp1*((nconfig(i,j)-1) - zsi(i,j)*u)
            else
               dRij = -Ceff(i,j)*zsi(i,j)*dexp(-zsi(i,j)*u)
            endif
            d2Rij=-zsi(i,j)*(tmp1 + dRij)
            if (nconfig(i,j) .gt. 2) then
               d2Rij=d2Rij + Ceff(i,j)*
     &              dble((nconfig(i,j)-1)*dble(nconfig(i,j)-2))*
     &              u**(nconfig(i,j)-3) * dexp(-zsi(i,j)*u)
            endif

            rhoi=rhoi + Rij
            drhoi=drhoi + dRij
            d2rhoi=d2rhoi + d2Rij
 20      continue
         rho=rho + Ns(i)*rhoi*rhoi/(4*3.1415927)
         drhodr=drhodr + 2.0*Ns(i)*rhoi*drhoi/(4*3.1415927)
         d2rhodr2=d2rhodr2 + 2.d0*Ns(i)*(drhoi*drhoi + rhoi*d2rhoi)
 10   continue
      return
      end

C-----------------------------------------------SUBROUTINE FSPLINEH-------
      subroutine fsplineH(u,F,dFdu,d2Fdu2)
C
C This subroutine calculates embedding energy using spline fit information
C
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of arguments
      double precision u, F, dFdu, d2Fdu2
```

```
C ...Declaration of local variables
      double precision x(MAXKNOTS),y(MAXKNOTS),u2

      integer nknots
      integer i

C Calculate Embedding Function for Host

      nknots=nspFH
      do 10 i=1,nknots
         x(i)=rhofH(i)
         y(i)=frhoH(i)
 10   continue

C Calculate Spline Coefficients
      if (.not. calcsplF) then
         call spline2(MAXKNOTS,nknots,x,y,d2FH)
         calcsplF=.true.
      endif

      if (u .gt. rhofH(nknots)) then
         u2=rhofH(nknots)
         call splint(MAXKNOTS,nknots,x,y,d2FH,u2,F,dFdu,d2Fdu2)
         F=frhoH(nknots) + dFdu*(u-u2)
         d2Fdu2=0.d0
      else
         u2=u
         call splint(MAXKNOTS,nknots,x,y,d2FH,u2,F,dFdu,d2Fdu2)
      endif

      return
      end

C-----------------------------------------------SUBROUTINE ZSPLINEH-------
      subroutine zsplineH(r,Z,dZdr,d2Zdr2)
C
C This subroutine calculates pairing energy using spline fit information
C
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of arguments
      double precision r, Z, dZdr, d2Zdr2

C ...Declaration of local variables
      double precision x(MAXKNOTS),y(MAXKNOTS)

      integer nknots
      integer i

C Calculate Pair Potential for Host

      nknots=nspZH
      do 10 i=1,nknots
         x(i)=rzH(i)
         y(i)=zrH(i)
 10   continue


C Calculate Spline Coefficients
      if (.not. calcsplZ) then
         call spline1(MAXKNOTS,nknots,x,y,d2ZH)
         calcsplZ=.true.
      endif
```

```
      if (r .gt. rzH(nknots)) then
         Z=0.d0
         dZdr=0.d0
      else
         call splint(MAXKNOTS,nknots,x,y,d2ZH,r,Z,dZdr,d2Zdr2)
      endif

      return
      end

C-----------------------------------------------SUBROUTINE ZANALH-------
      subroutine zanalH(r,Z,dZdr,d2Zdr2)
C
C This subroutine calculates embedding energy using analytical form
C Assumes that F is fitted by the following parameterized form:
C
C     Z(r)= (1 - r/2)**p
C
C This form fits the embedding energy calculated in
C  Puska et al.,PRB, vol. 24, 3037 (1981)
C
C UNITS:   r is in Angstroms
C
C
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of arguments
      double precision r, Z, dZdr,d2Zdr2


      Z=(1.d0 - r/2.d0)**pH
      dZdr=pH*(1.d0 - r/2.d0)**(pH-1.d0)
      d2Zdr2=pH*(pH-1.d0)*(1.d0 - r/2.d0)**(pH-2.d0)

      return
      end

C-----------------------------------------------SUBROUTINE FANALH-------
      subroutine fanalH(u,F,dFdu,d2Fdu2)
C
C This subroutine calculates embedding energy using analytical form
C Assumes that F is fitted by the following parameterized form:
C
C     F(u)= B1 * u + B2 + 1/(B3*u + B4)
C
C This form fits the embedding energy calculated in
C  Puska et al.,PRB, vol. 24, 3037 (1981)
C
C UNITS:   U is in  1/(Angstroms**3)
C
C
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of arguments
      double precision u, F, dFdu, d2Fdu2


      F= B1H*u + B2H + 1.d0/(B3H*u + B4H)
      dFdu=B1H - B3H/(B3H*u + B4H)**2
      d2Fdu2= B3H*B3H/(B3H*u + B4H)**3

      return
```

```
      end


C-----------------------------------------------SUBROUTINE ZANALI-------
      subroutine zanalI(r,Z,dZdr,d2Zdr2)
C
C This subroutine calculates embedding energy using analytical form
C Assumes that F is fitted by the following parameterized form:
C
C     Z(r)= (1 - r/2)**p
C
C This form fits the embedding energy calculated in
C  Puska et al.,PRB, vol. 24, 3037 (1981)
C
C UNITS:   r is in Angstroms
C
C
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of arguments
      double precision r, Z, dZdr,d2Zdr2


      if (r .lt. 2.d0) then
         Z=(1.d0 - r/2.d0)**pI
         dZdr=-pI*(1.d0 - r/2.d0)**(pI-1.d0)/2.0
         d2Zdr2=pI*(pI-1.d0)*(1.d0 - r/2.d0)**(pI-2.d0)
      else
         Z=0.d0
         dZdr=0.d0
         d2Zdr2=0.d0
      endif


      return
      end


C-----------------------------------------------SUBROUTINE FANALI-------
      subroutine fanalI(u,F,dFdu,d2Fdu2)
C
C This subroutine calculates embedding energy using analytical form
C Assumes that F is fitted by the following parameterized form:
C
C     F(u)= B1 * u + B2 + 1/(B3*u + B4)
C
C This form fits the embedding energy calculated in
C  Puska et al.,PRB, vol. 24, 3037 (1981)
C
C UNITS:   U is in  1/(Angstroms**3)
C
C
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C ...Declaration of arguments
      double precision u, F, dFdu, d2Fdu2


      F= B1I*u + B2I + 1.d0/(B3I*u + B4I)
      dFdu=B1I - B3I/(B3I*u + B4I)**2
      d2Fdu2= B3I*B3I/(B3I*u + B4I)**3
```

```
      return
      end


C-----------------------------------------------SUBROUTINE CALCRHOI------
      subroutine calcrhoI(r,rho,drhodr,d2rhodr2)
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C r is in angs.
C rho is in angs^-3. drhodr is in angs^-4

      double precision r,rho,drhodr,d2rhodr2

      double precision ab
      ab = 0.529
      rho = exp(-2.0*r/ab) / (pimath * (ab**3))
      drhodr = -2.0 * exp(-2.0*r/ab) / (pimath * (ab**4))

C     d2rhodr2 is broken. but i don't care about it since i am only
C     using gradient information
      d2rhodr2 = 0.0

      return
      end
```

```
C-----------------------------------------------------SUBROUTINE RDDATA------
      subroutine rddata(iunit,ndat,datarray)
C..declaration of arguments
      double precision datarray(50)
      integer iunit,ndat

C..declaration for local variables
      character*80 line,kstring
      integer nlist

      integer j

 10   continue
      read(iunit,101,err=900)line
 101  format(a80)

      if ((line(1:1) .ne. 'd') .and. (line(1:1) .ne. 'D')) goto 10
      kstring=line(2:)


      call parselin(kstring,ndat,datarray,nlist)


      if (ndat .lt. 0) then
         return
      else
         if (nlist .ne. ndat) then
            print *,'READ ERROR'
            print *,'----------'
            print *,kstring
            print *,ndat
            print *,nlist
            do 809 j=1,nlist
               print *,j,datarray(j)
 809        continue
            stop
         else
            return
         endif
      endif

 900  print *,'ERROR: FILE EMPTY'
      stop

      end


C-----------------------------------------SUBROUTINE PARSELIN -------
      subroutine parselin(kstring,nvalues,rlist,nlist)

C     ..declaration of arguments
      character*80 kstring
      double precision rlist(50),rword

      integer nvalues
      integer nlist

C     ..declaration of local variables
      character*80 temp,charword
      character*1 blank,comma,delimeter
      logical frstblnk

      integer ilen1,ilen2,jloc,ninputs,iloc,lnblnk
      integer j

      blank=' '
```

```
      comma=','
      delimeter=comma

      ilen1=lnblnk(kstring)

      temp=kstring(:ilen1)

C     Compress string to remove all extraneous blanks
      ilen2=ilen1
      jloc=1
      frstblnk=.false.
      do 2 j=1,ilen1
         if (temp(jloc:jloc) .eq. blank) then
            if (frstblnk) then
               temp(jloc:jloc)=comma
               frstblnk=.false.
               jloc=jloc+1
            else
               temp(jloc:)=temp(jloc+1:)
               ilen2=ilen2-1
            endif
         else
            if (temp(jloc:jloc) .eq. comma) then
               if (.not. frstblnk) then
                  temp(jloc:)=temp(jloc+1:)
                  frstblnk=.false.
               else
                  jloc=jloc+1
                  frstblnk=.false.
               endif
            else
               jloc=jloc+1
               frstblnk=.true.
            endif
         endif
 2    continue
      temp(ilen2+1:ilen2+1)=comma

C      print *,temp

      if (nvalues .lt. 0) then
         ninputs=50
      else
         ninputs=nvalues
      endif

      nlist=0
      if (ilen2 .eq. 0) then
         nlist=0
         rlist(1)=0.
         goto 30
      endif

C      print *,'****************************************'
 20   continue
C      print *,'------------------------------------'
      ilen2=lnblnk(temp)
C     print *,'a)',j,ilen2
      if ((ilen2 .eq. 0).or. (nlist .eq. nvalues)) goto 30
      if (delimeter .ne. blank) goto 15
 10   if (temp(:1) .eq. blank) then
         temp=temp(2:)
         goto 10
      endif
 15   continue
```

```
      iloc=index(temp,delimeter)
      charword=temp(1:iloc-1)
C      print *,'b)',temp
C      print *,'c)',charword
C      print *,'d)',j,iloc
      if (iloc .ne. 1) then
         read(unit=charword,fmt=*, err=25) rword
      else
         rword=0.d0
      endif
C      print *,'....(i)',rword
      nlist=nlist+1
      rlist(nlist)=rword
 25   temp=temp(iloc+1:)
C      print *,'f)',nlist
C      print *,'g)',temp
      goto 20

 30   continue
      return
      end

C-----------------------------------------------SUBROUTINE LOADDATA-----
      subroutine loaddata
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

      double precision dlist(50)
      integer itmp,iunit
      integer i,j



      open (unit=7,err=100,status='old',file='eamrun.dat')
      read(7,*)namhost
      read(7,*)namimp
      close(7)
      goto 120

 100  continue
      close(7)
      print *,'Host File Does Not Exist: Enter file name for Host data'
      read *,namhost
 101  continue
      print *,'Impurity File Does Not Exist: ',
     &     'Enter file name for Inp data'
      read *,namimp

      open (unit=8,file=namhost,err=100,status = 'old')
      close (8)
      open (unit=8,file=namimp,err=101,status = 'old')
      close (8)

      open (unit=7,err=100,file='eamrun.dat',status = 'old')
      rewind(7)
      write(7,*)namhost
      write(7,*)namimp
      close(7)

C...Read  In Host File Data

 120  continue
      open(unit=7,file=namhost,status = 'old')
      iunit=7
C     Read Lattice constant
```

```
      call rddata(iunit,1,dlist)
      a0=dlist(1)
      call rddata(iunit,1,dlist)
      a0fit=dlist(1)

C     Read Elastic Constants
      call rddata(iunit,3,dlist)
      c11=dlist(1)
      c12=dlist(2)
      c44=dlist(3)
      call rddata(iunit,3,dlist)
      c11fit=dlist(1)
      c12fit=dlist(2)
      c44fit=dlist(3)

C.. Read Sublimation Energy
      call rddata(iunit,1,dlist)
      Es=dlist(1)
      call rddata(iunit,1,dlist)
      Esfit=dlist(1)

C..Read Vacancy Formation Energy
      call rddata(iunit,1,dlist)
      Evacancy=dlist(1)
      call rddata(iunit,1,dlist)
      Evfit=dlist(1)

C..Read Embedding Energy Function
      call rddata(iunit,1,dlist)
      itmp=int(dlist(1))
      if (itmp .eq. 0) then
         analyH=.false.
      else
         analyH=.true.
      endif

      if (.not. analyH) then
C..Load spline data for host material...
         call rddata(iunit,1,dlist)
         nspFH=dlist(1)


         call rddata(iunit,1,dlist)
         rhobarH=dlist(1)

         do 20 i=1,nspFH
            call rddata(iunit,3,dlist)
            j=int(dlist(1))
            rhofH(i)=dlist(2)
            frhoH(i)=dlist(3)
 20      continue

C..Loadspline data for pair potential
         call rddata(iunit,1,dlist)
         nspZH=dlist(1)
         do 25 i=1,nspZH
            call rddata(iunit,3,dlist)
            j=int(dlist(1))
            rzH(i)=dlist(2)
            zrH(i)=dlist(3)
 25      continue
      else
C..Analytic Fit Parameters
         call rddata(iunit,5,dlist)
         B1H=dlist(1)
         B2H=dlist(2)
```

```
         B3H=dlist(3)
         B4H=dlist(4)
         pH=dlist(5)
      endif

C.....
C..........Read Atomic Density Parameters.....
C.....
      call rddata(iunit,1,dlist)
      Nshell=dlist(1)
      call rddata(iunit,Nshell,dlist)
      do 34 i=1,Nshell
         Ns(i)=dlist(i)
 34   continue
C..read in wavefunction from Clementi tables
      call rddata(iunit,1,dlist)
      densunit=dlist(1)
      do 38 i=1,Nshell
         call rddata(iunit,3,dlist)
         elevel(i)=int(dlist(1))
         angmom(i)=int(dlist(2))
         Norb(i)=int(dlist(3))

         do 40 j=1,Norb(i)
            call rddata(iunit,4,dlist)
            nconfig(i,j)=dlist(2)
            zsi(i,j)=dlist(3)
            C(i,j)=dlist(4)
 40      continue
 38   continue

      close(iunit)


C.....
C.....Read Impurity Data.....
C.....
      open(unit=7,file=namimp,status = 'old')
      iunit=7
      call rddata(iunit,1,dlist)
      B1I=dlist(1)
      call rddata(iunit,1,dlist)
      B2I=dlist(1)
      call rddata(iunit,1,dlist)
      B3I=dlist(1)
      call rddata(iunit,1,dlist)
      B4I=dlist(1)
      call rddata(iunit,1,dlist)
      pI=dlist(1)
      close(iunit)

      return
      end

C-----SUBROUTINE DISP1-----
      subroutine disp1
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

      print *,'Host Material:     ',namhost
      print *,'Impurity Material: ',namimp
      print *,' '
      write(*,800)'Quantity','Exp.','Fit'
      write(*,801)'1.  a0   :',a0,a0fit
      write(*,801)'2.  c11  :',c11,c11fit
```

```
      write(*,801)'3.  c12  :',c12,c12fit
      write(*,801)'4.  c44  :',c44,c44fit
      write(*,801)'5.  Es   :',Es,Esfit
      write(*,801)'6.  Ev   :',Evacancy,Evfit
      print *,' '

 800  format(a10,2x,a10,2x,a10)
 801  format(a10,2x,f10.4,2x,f10.4)

      return
      end

C-----------------------------------------------------SUBROUTINE DISP2-----
      subroutine disp2
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

      integer j

      if (.not. analyH) then
         print *,'---------------------------------'
         print *,'Host EAM Functions: Spline Fit'
         print *,'---------------------------------'
         print *,'Rhobar(Ang^(-3)) =',rhobarH*ang*ang*ang
         print *,' '
         write(*,804)'r','Z(r)'
         write(*,804)'-','----'
         do 790 j=1,nspZH
            write(*,805)rzH(j),zrH(j)
 790     continue
         print *,'  ...................'
         write(*,804)'rho','F(rho)'
         write(*,804)'---','------'
         do 791 j=1,nspFH
            write(*,805)rhofH(j),frhoH(j)
 791     continue
      else
         print *,'Host EAM Functions: Analytical'
         print *,'---------------------------------'
         print *,' '
         print *,'F(u)=B1*u + B2 + 1/(B3*u + B4)'
         print *,'Z(r)=(1 - r/2)^2'
         print *,' '
         write(*,806)'B1 =',B1H
         write(*,806)'B2 =',B2H
         write(*,806)'B3 =',B3H
         write(*,806)'B4 =',B4H
         write(*,806)'p =',pH
         print *,' '
      endif

      print *,' '
      print *,' '
      print *,'---------------------------------'
      print *,'Impurity EAM Functions: Analytical'
      print *,'---------------------------------'
      print *,' '
      print *,'F(u)=B1*u + B2 + 1/(B3*u + B4)'
      print *,'Z(r)=(1 - r/2)^2'
      print *,' '
      write(*,806)'B1 =',B1I
      write(*,806)'B2 =',B2I
      write(*,806)'B3 =',B3I
      write(*,806)'B4 =',B4I
      write(*,806)'p =',pI
```

```
      print *,' '


 804  format(a10,2x,a10)
 805  format(f10.4,2x,f10.4)
 806  format(a10,2x,f10.4)

      return
      end



C-----------------------------------------------SUBROUTINE DISP2-----
      subroutine disp3
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

      integer i,j

C..read in wavefunction from Clementi tables
      print *,'---------------------------------------'
      print *,'   Wave Functions from Clementi Tables '
      print *,'---------------------------------------'
      print *,'Units for zsi (in inverse angstroms: ',densunit/ang
      do 38 i=1,Nshell
         print *,'N=',elevel(i),' L=',angmom(i)
         do 40 j=1,Norb(i)
            write(*,801)j,nconfig(i,j),zsi(i,j),C(i,j)
 40      continue
         print *,' '
 38   continue

 801  format(i5,2x,i5,2x,f10.6,2x,f10.6)
      return
      end

C-----------------------------------------------SUBROUTINE OUTPUTF-------
      subroutine outputf
      include 'eamparam.com'
      include 'eamcom.def'
      include 'eamcom.dec'

C..declaration for local variables..
      double precision F1,F2,DF1,DF2,D2F1,D2F2
      double precision umax,umin,du,u,rho
      integer nu

      integer i
      nu=100

      print *,'Input  RHOMIN (units of RHO/RHOBAR)'
      read *,umin
      print *,'Input RHOMAX (units of RHO/RHOBAR)'
      read *,umax

      du=(umax - umin)/dfloat(nu-1)

      open(unit=7,file='fout.mat',status = 'new')
      u=umin

      do 20 i=1,nu
         rho=u*rhobarH
         call FH(rho,F1,DF1,D2F1)
         call FI(rho,F2,DF2,D2F2)
         write(7,801)i,u,F1/Es,F2/Es
```

```
         u=u+du
 20    continue
 801   format(i5,2x,f10.4,2x,f10.4,2x,f10.4)
       close(7)
       return
       end



C-------------------------------------------------SUBROUTINE OUTPUTZ-------
       subroutine outputz
       include 'eamparam.com'
       include 'eamcom.def'
       include 'eamcom.dec'

C..declaration for local variables..
       double precision Z1,Z2,DZ1,DZ2,D2Z1,D2Z2
       double precision umax,umin,du,u,r
       integer nu
       integer i

       nu=100

       print *,'Input  RMIN (Angstroms)'
       read *,umin
       print *,'Input RMAX (Angstroms)'
       read *,umax

       du=(umax - umin)/dfloat(nu-1)

       open(unit=7,file='zout.mat',status = 'new')
       u=umin

       do 20 i=1,nu
          r=u*ang
          call ZH(r,Z1,DZ1,D2Z1)
          call ZI(r,Z2,DZ2,D2Z2)
          write(7,801)i,u/a0,Z1,Z2
          u=u+du
 20    continue
       close(7)

 801   format(i5,2x,f10.4,2x,f10.4,2x,f10.4)

       return
       end


C-------------------------------------------------SUBROUTINE OUTPUTN-------
       subroutine outputn
       include 'eamparam.com'
       include 'eamcom.def'
       include 'eamcom.dec'

C..declaration for local variables..
       double precision N1,DN1,DN2
       double precision umax,umin,du,u,r
       integer nu
       integer i

       nu=1000

       print *,'Input  RMIN (Angstroms)'
       read *,umin
       print *,'Input RMAX (Angstroms)'
       read *,umax
```

```
      if (umin .lt. 0.d0) umin=0.d0

      du=(umax - umin)/dfloat(nu-1)

      open(unit=7,file='nout.mat',status = 'new')
      u=umin

      do 20 i=1,nu
         r=u*ang
         call calcrho(r,N1,DN1,DN2)
         write(7,801)i,u,N1,DN1,DN2
         u=u+du
20    continue

      close(7)
801   format(i5,2x,f10.4,2x,f10.4,2x,f14.4,2x,f14.4)

      return
      end
```

```
integer MAXKNOTS,NT,MT
parameter(MAXKNOTS=20)
parameter(NT = 18)
parameter(MT = 153)
```

```
"PdHost.dat"
 "DImp.dat"
```

```
      double precision function func(DynX)

      external CalcPhi,CalcF
      include 'eamparam.com'
      include 'hfparamPert.com'
      include 'eamcom.def'
      include 'hfcomPert.def'
      include 'eamcom.dec'
      include 'hfcomPert.dec'

      double precision CalcPhi,CalcF
      double precision DynX(NDynX*3)
      double precision PHI,F
      integer i

C     convert to the supercell format

C      print *,'IN FUNC'

      do 10 i=1,NDynX
         SuperCell(i,1) = DynX((i-1)*3 + 1)
         SuperCell(i,2) = DynX((i-1)*3 + 2)
         SuperCell(i,3) = DynX((i-1)*3 + 3)
 10   continue

      call NNeighbour()

C     print *,'calling CalcPhi'
      PHI = CalcPhi()
C     print *,'called CalcPhi'
C      print *,'calling CalcF'
      F = CalcF()
C     print *,'called CalcF'

      print *,'PHI,F'
      PHI = PHI*1.0d+10
      F = F*1.0d+10
      print *,PHI,F
      func = (PHI + F)
C     print *,'LEAVING FUNC'
      return
      end

C     ----------------------------------------------------------------

      double precision function CalcPhi()

      include 'eamparam.com'
      include 'hfparamPert.com'
      include 'eamcom.def'
      include 'hfcomPert.def'
      include 'eamcom.dec'
      include 'hfcomPert.dec'

      double precision r
      double precision z1,dz1dr,d2z1dr2
      double precision z2,dz2dr,d2z2dr2
      integer Type1,Type2
      integer i,j


C     calculate the pair potential by going only to the nearest
C     neighbours

C      print *,'IN CALCPHI'
      Phi = 0.0
```

```
      do 20 i=1,NASC
         Type1 = TypeSC(i)
         do 30 j=1,NSNN(i)
            Type2 = TypeSC(SList(i,j))
            r = sqrt(SNN(i,j,1)**2 + SNN(i,j,2)**2 + SNN(i,j,3)**2)
            if (Type1.eq.1 .and. Type2.eq.1) then
               call  ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               Phi = Phi + z1*z1/(r*ang)
            elseif ((Type1.eq.1 .and. Type2.eq.46) .or. (Type1.eq.46
     *              .and.Type2.eq.1)) then
               call ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               call ZH(r*ang,z2,dz2dr,d2z2dr2)
               z2 = z2*e
               Phi = Phi + z1*z2/(r*ang)
            elseif (Type1.eq.46 .and. Type2.eq.46) then
               call ZH(r*ang,z1,dz1dr,d2z1dr2)
C               print *,Type1,Type2
C               print *,'Type1,Type2'
C               print *,'r,z1,dz1dr'
C               print *,r,z1,dz1dr
               z1 = z1*e
               Phi = Phi + z1*z1/(r*ang)
            else
               print *,'Type1,Type2'
               print *,Type1,Type2
               print *,'error in CalcPhi1'
               stop
            endif
 30      continue

         do 40 j=1,NDNN(i)
            Type2 = TypeSC(DList(i,j))
            r = sqrt(DNN(i,j,1)**2 + DNN(i,j,2)**2 + DNN(i,j,3)**2)

            if (Type1.eq.1 .and. Type2.eq.1) then
               call  ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               Phi = Phi + z1*z1/(r*ang)
            elseif ((Type1.eq.1 .and. Type2.eq.46) .or. (Type1.eq.46
     *              .and.Type2.eq.1)) then
               call ZI(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               call ZH(r*ang,z2,dz2dr,d2z2dr2)
               z2 = z2*e
               Phi = Phi + z1*z2/(r*ang)
            elseif (Type1.eq.46 .and. Type2.eq.46) then
               call ZH(r*ang,z1,dz1dr,d2z1dr2)
               z1 = z1*e
               Phi = Phi + z1*z1/(r*ang)
            else
               print *,'Type1,Type2'
               print *,Type1,Type2
               print *,'error in CalcPhi2'
               stop
            endif
 40      continue
 20   continue

      CalcPhi = Phi/2.0
C      print *,Phi/2.0
C      print *,'LEAVING CALCPHI'
      return
      end
```

```
C     ---------------------------------------------------------------

      double precision function CalcF()
      include 'eamparam.com'
      include 'hfparamPert.com'
      include 'eamcom.def'
      include 'hfcomPert.def'
      include 'eamcom.dec'
      include 'hfcomPert.dec'

      double precision r
      double precision rho,drhodr,drho2dr2,TotalRho
      double precision F,dFdrho,d2Fdrho2,TotalF
      integer i,j,Type1,Type2

C      print *,'IN CALCF'
      TotalF = 0.0
      F = 0.0
      do 10 i=1,NASC
         TotalRho = 0.0
         Type1 = TypeSC(i)
         do 20 j=1,NSNN(i)
            Type2 = TypeSC(SList(i,j))
            r = sqrt(SNN(i,j,1)**2 + SNN(i,j,2)**2 + SNN(i,j,3)**2)
            if (Type2.eq.1) then
               call calcrhoI(r,rho,drhodr,drho2dr2)
            elseif(Type2.eq.46) then
               call calcrho(r*ang,rho,drhodr,drho2dr2)
C               print *,'r,rho'
C               print *,r,rho,1
            else
               print *,'error'
               stop
            endif
            TotalRho = TotalRho + rho
 20      continue

         do 30 j=1,NSSNN(i)
            Type2 = TypeSC(SList1(i,j))
            r = sqrt(SSNN(i,j,1)**2 + SSNN(i,j,2)**2 + SSNN(i,j,3)**2)
            if (Type2.eq.1) then
               call calcrhoI(r,rho,drhodr,drho2dr2)
            elseif(Type2.eq.46) then
               call calcrho(r*ang,rho,drhodr,drho2dr2)
C               print *,'r,rho'
C               print *,r,rho,2
            else
               print *,'error'
               stop
            endif
            TotalRho = TotalRho + rho
 30      continue

         do 35 j=1,NSTNN(i)
            Type2 = TypeSC(SList2(i,j))
            r = sqrt(STNN(i,j,1)**2+STNN(i,j,2)**2+STNN(i,j,3)**2)
            if (Type2.eq.1) then
               call calcrhoI(r,rho,drhodr,drho2dr2)
            elseif(Type2.eq.46) then
               call calcrho(r*ang,rho,drhodr,drho2dr2)
C               print *,'r,rho'
C               print *,r,rho,1
            else
               print *,'error'
               stop
            endif
```

```
            TotalRho = TotalRho + rho
 35       continue


          do 40 j=1,NDNN(i)
             Type2 = TypeSC(DList(i,j))
             r = sqrt(DNN(i,j,1)**2 + DNN(i,j,2)**2 + DNN(i,j,3)**2)
             if (Type2.eq.1) then
                call calcrhoI(r,rho,drhodr,drho2dr2)
             elseif(Type2.eq.46) then
                call calcrho(r*ang,rho,drhodr,drho2dr2)
             else
                print *,'error'
                stop
             endif
             TotalRho = TotalRho + rho
 40       continue

          if (Type1.eq.1) then
             call FI(TotalRho*densunit*densunit*densunit,F,dFdrho,
     *            d2Fdrho2)
             F = F*eV
          elseif (Type1.eq.46) then
             call FH(TotalRho*densunit*densunit*densunit,F,dFdrho,
     *            d2Fdrho2)
             F = F*eV
C             print *,'TotalRho'
C             print *,TotalRho
C             print *,'F'
C             print *,F
          else
             print *,'error in CalcF'
             stop
          endif
          TotalF = TotalF + F
 10    continue

C       print *,'TotalF'
C       print *,TotalF
        CalcF = TotalF
C       print *,'LEAVING CALCF'
        return
        end

C       ----------------------------------------------------------------
```

```
C       a0 is the lattice constant.
C       DM is the Dynamical Matrix
C       l is the array of lattice vectors
C       NSNN(i) is the no. of nearest neighbours of i of type i
C       NDNN(i) is the no. of nearest neighbours of i of type not i
C       SNN(i,j,a) labels self nearest neighbours. i is the atom under
C       consideration, j is the no. of nearest neighbour, and a is the
C       x,y,z relative coordinate of the nearest neighbour. 12 is
C       used since the max. no. of nearest neighbours of any atom
C       can be 12.
C       Similarly for DNN except it labels atoms i's nearest neighbours
C       whose type is different from atom i.
C       SuperCell contains the numbering and coordinates of SuperCell
C       TypeSC contains the no. and type of atoms in SuperCell
C       SList(i,j) is the self list of nearest neighbours of atom i
C       (i is the no. in the SuperCell)
C       and nearest neigbour j(from 1-12) and it contains the no. of
C       the nearest neighbour j in SuperCell, and thus helps to form
C       the dynamical matrix. Similarly for DList
C       SDistCell(alpha,beta,i) gives the coordinates of the difference
C       between the cells in which alpha and beta live. i is the cartesian
C       coordinate.alpha is always in the supercell, whereas beta can
C       be in the supercell or in the neighbouring cells.
C       TDegenerate measures the times degenerate perturbation
C       theory was called.

        double precision l(3,3)
        integer NSNN(NASC),NDNN(NASC)
        double precision SNN(NASC,24,3),DNN(NASC,24,3)
        integer SList(NASC,24),DList(NASC,24)
        integer TypeSC(NASC)
        double precision SuperCell(NASC,3)
C       double precision SDistCell(NASC,24,3),DDistCell(NASC,24,3)

        integer NSSNN(NASC)
        double precision SSNN(NASC,24,3)
        integer SList1(NASC,24)

        integer NSTNN(NASC)
        double precision STNN(NASC,50,3)
        integer SList2(NASC,50)

        double precision TDegenerate
```

```
common /lattice/l
common /Dmatrix1/NSNN,NDNN,SList,DList
common /Dmatrix2/SNN,DNN
common/Scell1/TypeSC
common/Scell2/SuperCell
common/DmatrixSNN1/SSNN,STNN
common/DmatrixSNN2/SList1,NSSNN
common/DmatrixSNN3/SList2,NSTNN
common/totalDegen/TDegenerate
```

```
C       NASC is the no. of atoms in the supercell.
C       epsilon checks equailities to within that precision
        integer NASC
        double precision epsilon,MPd,MD,Pi
        complex*16 iota
        PARAMETER (NASC = 2)
        PARAMETER (epsilon = 0.001)
        PARAMETER (MPd = 106.90348)
        PARAMETER (MD = 2.014102)
        PARAMETER (Pi = 3.141593)
        PARAMETER (iota = (0.0,1.0))
```

```
C       NASC is the no. of atoms in the supercell.
C       epsilon checks equailities to within that precision
        integer NASC,NDynX
        double precision epsilon,MPd,MD
        complex*16 iota
        PARAMETER (NASC = 53)
        PARAMETER (NDynX = 18)
        PARAMETER (epsilon = 0.001)
        PARAMETER (MPd = 106.90348)
        PARAMETER (MD = 1.00727)
        PARAMETER (iota = (0.0,1.0))
```

```
      integer a1(3),a2(3),a3(3)
      integer tempPd(3),tempD(3)
      integer i,j,k,alpha
      double precision r1,r2

      a1(1) = 10
      a1(2) = 10
      a1(3) = 0

      a2(1) = 10
      a2(2) = 0
      a2(3) = 10

      a3(1) = 0
      a3(2) = 10
      a3(3) = 10


      alpha = 0
      do 10 i=0,1
         do 20 j=0,1
            do 30 k=0,1
               alpha = alpha + 1
               tempPd(1) = i*a1(1) + j*a2(1) + k*a3(1)
               tempPd(2) = i*a1(2) + j*a2(2) + k*a3(2)
               tempPd(3) = i*a1(3) + j*a2(3) + k*a3(3)
               tempD(1) = tempPd(1) + 10
               tempD(2) = tempPd(2)
               tempD(3) = tempPd(3)
               print *,tempPd(1),tempPd(2),tempPd(3),11,-46,11
               print *,tempD(1),tempD(2),tempD(3),11,-1,11

30          continue
20       continue
10    continue
      end
```

```
      subroutine main3

      include 'eamparam.com'
      include 'hfparamPert.com'
      include 'eamcom.def'
      include 'hfcomPert.def'
      include 'eamcom.dec'
      include 'hfcomPert.dec'


      double precision ftol,fret
      integer iter,i,j
      double precision DynX(3*NDynX)

C      print *,'IN MAIN'

      l(1,1) = 1.5*a0
      l(1,2) = 1.5*a0
      l(1,3) = 0.0

      l(2,1) = 1.5*a0
      l(2,2) = 0.0
      l(2,3) = 1.5*a0

      l(3,1) = 0.0
      l(3,2) = 1.5*a0
      l(3,3) = 1.5*a0

      ftol = 1.0d-4
      call LoadDataM(DynX)
      call NNeighbour()
      print *,'calling frprmn'
      call frprmn(DynX,3*NDynX,ftol,iter,fret)
      open (unit = 7, file = 'temp.cell')
      do 100 i=1,NDynX
         write (7,*) DynX((i-1)*3+1),DynX((i-1)*3+2),
     *        DynX((i-1)*3+3),11,-TypeSC(i),11
 100  continue
      close(7)

      do 200 i=1,3*NDynX
         print *,DynX(i)
 200  continue
      print *,fret
      print *,'called frprmn'
C     print *,'LEAVING MAIN'
      return
      end

C     ------------------------------------------------------------
C     this subroutine calculates the nearest neighbours of the atoms
C     in the supercell

      subroutine NNeighbour()

      include "eamparam.com"
      include "hfparamPert.com"
      include 'eamcom.dec'
      include "hfcomPert.dec"
      include 'eamcom.def'
      include "hfcomPert.def"

C     declare local variables
      integer alpha,beta,i,j,k,TypeA,TypeNN
      double precision DistA1,DistA2,DistA3,SelfNNDist,DifDist,distance
      double precision DispX,DispY,DispZ,DispBeta(3)
```

```
      double precision SelfSNNDist

      DistA1 = sqrt(l(1,1)**2 + l(1,2)**2 + l(1,3)**2)
      DistA2 = sqrt(l(2,1)**2 + l(2,2)**2 + l(2,3)**2)
      DistA3 = sqrt(l(3,1)**2 + l(3,2)**2 + l(3,3)**2)

C     SelfNND = 0.7, SelfSNND = 1.0, DifNNDist = 0.5,
C     DifSNNdist= sqrt(1.5) = 1.12
C     So break points are chosen half way in between

      SelfNNDist = 0.85*a0
      SelfSNNDist = 1.15*a0
      DifDist = 0.80*a0
      do 10 alpha=1,NASC
         TypeA = TypeSC(alpha)
         NSNN(alpha) = 0
         NDNN(alpha) = 0
         NSSNN(alpha) = 0
         do 20 beta=1,NASC
            TypeNN = TypeSC(beta)
            do 30 i=-2,2
               do 40 j=-2,2
                  do 50 k=-2,2

                     DispX=dble(i)*l(1,1)+dble(j)*l(2,1)+dble(k)*l(3,1)
                     DispY=dble(i)*l(1,2)+dble(j)*l(2,2)+dble(k)*l(3,2)
                     DispZ=dble(i)*l(1,3)+dble(j)*l(2,3)+dble(k)*l(3,3)

                     DispBeta(1) = DispX + SuperCell(beta,1)
                     DispBeta(2) = DispY + SuperCell(beta,2)
                     DispBeta(3) = DispZ + SuperCell(beta,3)

                     distance =
     *                    (SuperCell(alpha,1)-DispBeta(1))**2+
     *                    (SuperCell(alpha,2)-DispBeta(2))**2+
     *                    (SuperCell(alpha,3)-DispBeta(3))**2
                     distance = sqrt(distance)
                     if(TypeNN .eq. TypeA) then
                        if ((distance .gt. (SelfSNNDist)).or.
     *                       (distance .lt. epsilon)) then
                           continue

                        elseif ((distance .gt. (SelfNNDist))
     *                          .and. (distance .lt. (SelfSNNDist
     *                          ))) then
                           NSSNN(alpha) = NSSNN(alpha)+1
                           SList1(alpha,NSSNN(alpha)) = beta
                           SSNN(alpha,NSSNN(alpha),1) =
     *                          DispBeta(1) - SuperCell(alpha,1)
                           SSNN(alpha,NSSNN(alpha),2) =
     *                          DispBeta(2) - SuperCell(alpha,2)
                           SSNN(alpha,NSSNN(alpha),3) =
     *                          DispBeta(3) - SuperCell(alpha,3)

                        elseif (distance .lt. (SelfNNDist))
     *                          then
                           NSNN(alpha) = NSNN(alpha)+1
                           SList(alpha,NSNN(alpha)) = beta
                           SNN(alpha,NSNN(alpha),1) =
     *                          DispBeta(1) - SuperCell(alpha,1)
                           SNN(alpha,NSNN(alpha),2) =
     *                          DispBeta(2) - SuperCell(alpha,2)
                           SNN(alpha,NSNN(alpha),3) =
     *                          DispBeta(3) - SuperCell(alpha,3)

                        else
```

```
                        print *,'error in self nearest neighbour'
                     endif

                  else

C     so they are different

                     if (distance .gt. (DifDist)) then
                        continue

                     elseif (distance .lt. (DifDist)) then
                        NDNN(alpha) = NDNN(alpha)+1
                        DList(alpha,NDNN(alpha)) = beta
                        DNN(alpha,NDNN(alpha),1) =
     *                       DispBeta(1) - SuperCell(alpha,1)
                        DNN(alpha,NDNN(alpha),2) =
     *                       DispBeta(2) - SuperCell(alpha,2)
                        DNN(alpha,NDNN(alpha),3) =
     *                       DispBeta(3) - SuperCell(alpha,3)

                     else
                        print *,'error in dif. nearest neighbour'
                     endif
                  endif
 50            continue
 40         continue
 30      continue
 20    continue
 10    continue
       return
       end

C     ------------------------------------------------------------------
C     this subroutine loads the data

      subroutine LoadDataM(DynX)

      include "hfparamPert.com"
      include 'eamparam.com'
      include "hfcomPert.dec"
      include 'eamcom.dec'
      include "hfcomPert.def"
      include 'eamcom.def'

      double precision DynX(NDynX*3)

C     declare local variables
      integer i,dum1,dum2,type
      integer x,y,z

      double precision x1,y1,z1

      open(7,file = 'dyn.cell')
      do 10 i=1,NDynX
         read (7,*) x1,y1,z1,dum1,type,dum2
         SuperCell(i,1) = x1
         SuperCell(i,2) = y1
         SuperCell(i,3) = z1

         DynX((i-1)*3 + 1) = x1
         DynX((i-1)*3 + 2) = y1
         DynX((i-1)*3 + 3) = z1

         TypeSC(i) = -type
 10    continue
       close(7)
```

```
      open(7,file = 'stat.cell')
      do 20 i = NDynX+1, NASC

         read (7,*) x,y,z,dum1,type,dum2
C     The cell data comes in with a lattice constant of 20. we need
C     to divide by 20 to get it to a lattice constant of 1.

         SuperCell(i,1) = x*dble(a0)/20.0
         SuperCell(i,2) = y*dble(a0)/20.0
         SuperCell(i,3) = z*dble(a0)/20.0
         TypeSC(i) = -type

 20   continue
      close(7)
      return
      end
```

```
C------------------------------------------------ SUBROUTINE SPLINE1 ------
      subroutine spline1(MAXKNOTS,n,x,y,y2)
      integer NMAXF
      parameter(NMAXF=100)

C ..declaration of arguments
      double precision x(MAXKNOTS),y(MAXKNOTS),y2(NMAXF),u(NMAXF)
      integer n,MAXKNOTS

C ..declaration of local variables
      double precision sig,p,un,qn
      integer i,k

      if (n .gt. NMAXF) then
         print *,'ERROR (SPLINE2): N > NMAXF'
         stop
      endif

      y2(1)=-0.5
      u(1)=(3./(x(2)-x(1)))*(y(2)-y(1))/(x(2)-x(1))

      do 11 i=2,n-1
        sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))
        p=sig*y2(i-1)+2.d0
        y2(i)=(sig-1.)/p
        u(i)=(6.d0*((y(i+1)-y(i))/(x(i+1)-x(i))-(y(i)-y(i-1))
     *      /(x(i)-x(i-1)))/(x(i+1)-x(i-1))-sig*u(i-1))/p
11    continue

      qn=.5
      un=-(3./(x(n)-x(n-1)))*(y(n)-y(n-1))/(x(n)-x(n-1))

      y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.d0)
      do 12 k=n-1,1,-1
        y2(k)=y2(k)*y2(k+1)+u(k)
12    continue
      return
      end

C------------------------------------------------ SUBROUTINE SPLINE2 ------
      subroutine spline2(MAXKNOTS,n,x,y,y2)
      integer NMAXF
      parameter(NMAXF=100)

C ..declaration of arguments
      double precision x(MAXKNOTS),y(MAXKNOTS),y2(NMAXF),u(NMAXF)
      integer n,MAXKNOTS

C ..declaration of local variables
      double precision sig,p,un,qn
      integer i,k

      if (n .gt. NMAXF) then
         print *,'ERROR (SPLINE2): N > NMAXF'
         stop
      endif

      y2(1)=0.d0
      u(1)=0.d0

      do 11 i=2,n-1
        sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))
        p=sig*y2(i-1)+2.d0
        y2(i)=(sig-1.)/p
        u(i)=(6.d0*((y(i+1)-y(i))/(x(i+1)-x(i))-(y(i)-y(i-1))
     *      /(x(i)-x(i-1)))/(x(i+1)-x(i-1))-sig*u(i-1))/p
```

```
11    continue

      qn=0.d0
      un=0.d0

      y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.d0)
      do 12 k=n-1,1,-1
        y2(k)=y2(k)*y2(k+1)+u(k)
12    continue
      return
      end

C---------------------------------------------- SUBROUTINE SPLINT -------
      subroutine splint(MAXKNOTS,n,xa,ya,y2a,x,y,dydx,d2ydx2)

C ..declaration of argument
      double precision xa(MAXKNOTS),ya(MAXKNOTS),y2a(MAXKNOTS)
      double precision x,y,dydx,d2ydx2
      integer n,MAXKNOTS

C ..declaration of local variables
      double precision h,a,b

      integer klo,khi,k

      klo=1
      khi=n
1     if (khi-klo.gt.1) then
        k=(khi+klo)/2
        if(xa(k).gt.x)then
          khi=k
        else
          klo=k
        endif
      goto 1
      endif
      h=xa(khi)-xa(klo)
      if (h.eq.0.) pause 'bad xa input.'
      a=(xa(khi)-x)/h
      b=(x-xa(klo))/h
      y=a*ya(klo)+b*ya(khi)+
     *      ((a**3-a)*y2a(klo)+(b**3-b)*y2a(khi))*(h**2)/6.d0
      dydx=(ya(khi)-ya(klo))/h  - (3.d0*a*a -1.d0)*h*y2a(klo)/6.d0 +
     &     (3.d0*b*b - 1.d0)*h*y2a(khi)/6.d0
      d2ydx2=a*y2a(klo) + b*y2a(khi)

      return
      end
```