MOLECULAR DYNAMICS SIMULATION OF SILICON USING EMPIRICAL TIGHT-BINDING METHOD

BY

Ahmet Isik

Bachelor of Science in Nuclear Engineering
University of Hacettepe, Ankara, TURKEY, 1987

SUBMITTED TO THE DEPARTMENT OF NUCLEAR ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

AT THE

Massachusetts Institute of Technology

JANUARY 1992

Signature of Author_____

Department of Nuclear Engineering
27 January 1992

Certified by_____

Professor Sidney Yip
Department of Nuclear Engineering
Thesis Supervisor

Accepted by_____

Professor Allan F. Henry
Professor of Nuclear engineering
Chairman, Departmental Committee on Graduate Students

MOLECULAR DYNAMICS SIMULATION OF SILICON USING EMPIRICAL TIGHT-BINDING METHOD

BY

Ahmet Isik

Submitted to the Department of Nuclear Engineering on 27 January 1992 in the partial fulfillment of the requirements of the degree of

Master of Science

at the

Massachusetts Institute of Technology

ABSTRACT

An empirical tight-binding approximation used by Wang, Chan, and Ho (Phys. Rev. B, 39:8586,1988) to calculate interatomic forces in a molecular dynamics (MD) simulation code proved to be very fruitful in predicting certain physical properties of Si. In this thesis, this approach has been rederived and implemented in a set of codes to investigate materials properties of Si.

The electronic wavefunction of the system was constructed as a linear combination of s and p type atomic orbitals and the Hamiltonian matrix was constructed by using Harrison's $1/r^2$ law. The Hellman-Feynman forces were calculated by diagonalization of the Hamiltonian matrix. The repulsive short range pair potential was constructed by subtracting the band-structure energy per atom from the total energy of Si per atom which was obtained from literature.

The electronic degrees of freedom were explicitly used in the calculations of interatomic forces, so that our method is superior to those methods in which interatomic forces are calculated by using empirical potential functions.

Even though ab-initio methods permit more realistic descriptions of the atomic systems, the heavy computational cost of calculations severely restricts the number of particles contained in the system of MD simulation, so that only those materials phenomena which require involvement of small numbers of atoms can be investigated by MD simulation. But the computa-

tional requirements of empirical tight-binding method are moderate enough to permit simulation of systems containing particles up to 1000, so that many interesting materials phenomena can be investigated with a reasonably realistic description of the system.
The present set of codes has been validated by calculating the band-structure energy, the specific heat, and the radial distribution function of the system and by comparing our results with the ones obtained from the literature. Good agreement was obtained between our results and the ones in the literature. The total energy was conserved to the fifth digit.

Thesis Supervisor: **Sidney Yip**
Title: Professor of Nuclear Engineering

ACKNOWLEDGEMENT

My sincere thanks to my advisor Professor Sidney Yip for his patient help and guidance in completing this thesis.

I also would like to thank my colleague Meijie Tang for her friendship and the warm working atmosphere she created.

My deepest gratitude is to my parents for everything good they have done for me.

This research was supported by the Ministry of Education of the Government of the Republic of Turkey. I gratefully acknowledge their support during my thesis work.

Contents

1.1 Introduction 7

2 Interatomic Interactions **10**

2.1 The Many-Body Problem 10

2.2 Density Functional Approach 11

2.3 The Car-Parrinello Method 12

2.4 Empirical Potential Functions 12

2.5 The Empirical Tight-Binding Approximation 14

3 Derivation of Empirical Tight-Binding Approximation **17**

3.1 The System 17

3.2 Schrödinger Equation 19

3.3 Adiabatic Approximation 23

3.4 One-Electron Approximation 25

3.5 The Linear Combination of Atomic Orbitals 28

3.6 Construction of Repulsive Pair Potential 31

3.7 Tight-Binding Parameters 32

4 Molecular Dynamics Simulation **34**

4.1 The Code 34

4.2 Results and Discussions 41

4.3 Conclusions 48

List of Figures

4.1 Flowchart of code QUANTUM. 35

4.2 Total potential energy of Si in diamond structure as a function of the nearest-neighbor distance. 36

4.3 Band-structure energy per atom of silicon in diamond structure calculated by code QUANTUM. 38

4.4 The band-structure energy per silicon atom in the diamond-cubic structure taken from literature. 39

4.5 The repulsive pair potential per silicon atom in diamond structure. 40

4.6 The total energy of Si per atom as a function of temperature for constant volume case. 41

4.7 The total energy of Si per atom as a function of temperature for constant volume case taken from literature. 42
4.8 g(r) for T=100K 43
4.9 g(r) for T=200K 44
4.10 g(r) for T=300K 45
4.11 g(r) for T=500K 46
4.12 g(r) for T=600K 47

1.1 Introduction

Molecular dynamics (MD) simulation by using appropriate interatomic potentials provides us with unprecedented opportunities to understand materials behavior.[1] Not only can many materials properties be obtained by using the MD data, but also experiments which cannot be performed in the laboratory environment can be carried out easily on the model system. MD simulation also serves as a testing ground for the solid state theories and statistical mechanics.

What is essentially done is to find the phase trajectory of a given system of particles (atoms,molecules,ions..) and to calculate various expectation values

of the materials properties by using the principles of statistical mechanics. To find the phase trajectory of the system (coordinates and velocities of each paticle at each time step), we need to integrate the equations of motion of the particles. The solution of the equations of motion requires us to find the net forces acting on particles. Calculation of the forces is the most vital part of the MD simulation. The results of the simulation can be expected to represent the actual system only if the interactions between the particles represent the interactions in the real system.

The interactions among particles have to be known explicitly to carry out the MD simulation. The forces in the MD simulation are of quantum mechanical character. By using the rules of quantum mechanics, except for nuclear and gravitational forces, we can calculate all forces in the nature. For MD simulation we need to solve the Schrödinger equation for a many body system. The exact solution of the Schrödinger equation is possible only for a few extremely simple cases, like the harmonic oscillator or the hydrogen atom. For a system as simple as the helium atom, we do not have an exact solution of the Schrödinger equation, though by using numerical methods, a solution close to the exact solution can be obtained.

This thesis is concerned with the MD simulation of silicon in the diamond

structure by using the empirical tight-binding approximation[2] to the solution of the Schrödinger equation. In this approximation, we are in effect solving the many-body interaction problem among silicon nuclei and electrons, so that the method is superior to the ones which represent the interatomic interactions as a sum of few body terms, like the Stillinger-Weber potential [3, 4, 5] constructed to represent the interactions among silicon atoms. In the present work, we will show that the tight binding approximation is able to describe silicon in the diamond structure at high temperatures. We will show that silicon in diamond structure is indeed stable at high temperatures. This result is significant, because we know that no pair-potential can stabilize silicon in the diamond structure. To test the empirical tight-binding method further, we also calculated the specific heat of silicon in diamond structure at constant volume which agreed very well with the calculations previously reported.[6].

Chapter 2

Interatomic Interactions

2.1 The Many-Body Problem

The ideal problem we need to solve in order to represent the many-body interactions perfectly is the time-dependent Schrödinger equation for the system of the silicon nuclei plus the electrons. The interactions among these particles are Coulombic and the potential energy of two charged particles depends on the inverse of the distance between the two particles. We can write the Hamiltonian of the system which corresponds to the total energy of the system in the classical approximation.

Because the solution of the full Schrödinger equation for a system of many

particles is out of the question with today's computers, some approximations must be made to the Schrödinger equation. The Schrödinger equation cannot be solved exactly even for a single Si atom. The Hartree-Fock approximation which is derived below assumes that an electron is moving in the average field created by all the other electrons and the ionic cores and tries to find the best one-electron wave functions as the solution to the self-consistent Hartree-Fock equation. But the determination of the self-consistent solutions is computationally very demanding, and the application of this method to the MD simulation of atomic systems is for now out of the question.

2.2 Density Functional Approach

The density functional theory[7, 8] simplifies the Hartree-Fock equation and enables us to solve the Schrödinger equation for larger systems. In this approximation, instead of looking for the functions which solves the HF equation self consistently, we just need to find the eigenvalues of the Hamiltonian matrix. Because finding numbers is much easier than finding functions, the density functional approximation is computationally less expensive to deal with than the HF equation.

2.3 The Car-Parrinello Method

The Car-Parrinello method[9, 10, 11, 12] is an ab-initio method which was used successfully in the MD simulation of small atomic systems. By using the adiabatic approximation which is described below and and the local density approximation,[13] they were able to solve the equations of motion of both electrons and the cores simultaneously.[14]. But this method is computationally too expensive and the number of particles which can be simulated is on the order of one hundred or fewer. Unfortunately, many important materials phenomena, like the dislocation formation, requires simulation of many more atoms, so the Car-Parrinello method is unsuitable for the investigation of those phenomena with today's computing capabilities. But the development of new computers can expand the applications of the ab-initio methods to larger systems as well.

2.4 Empirical Potential Functions

The first potential functions which were used in the MD simulation of materials were empirical ones which mimiced the quantum mechanical effects. Those potentials are still of importance because they enable us to simulate

large systems (tens of thousands of atoms) compared to the quantum mechanical approaches. For example,the Lennard-Jones 6-12 potential function[15] gives good results in the simulation of inert gases, although the discrepancy between the simulation data and the experimental data increases with the atomic number of the ideal gas.

We can trace the origin of this potential to the interaction of induced dipoles created by the fluctation of electron density in the neutral atoms.

Another popular method which is used in the MD simulations of metals is the Embedded Atom Method(EAM)[16, 17]. The reason for the success of this method is its ability to include the effects of the electron density around cores in the calculation of the interatomic forces. For metals, these environmental effects are important contributors to the interatomic potential. The electron density around an ion must be calculated and included in the calculation of forces.

The forces among atoms in covalent systems are strongly direction-dependent and cannot be calculated with pair potentials. It is known that no pair potential can stabilize the tetrahedral structure we observe in silicon, diamond and many other technologically important materials. At least the three-body effects must be taken into account. The "Stillinger-Weber" (SW)

potential[3, 4, 5] stabilizes the tetrahedral structure by including angular terms besides two-body terms. The origin of this potential can be again traced to the quantum-mechanical structure of the system.[18]

The advantage of the empirical potential functions is that they can be used easily in the MD simulations with considerable savings in computation. For example, the hard-sphere potential which is one of the simplest potential functions enables us to simulate literally millions of particles. No real atomic systems obey such a simple potential function, but the hard-spheres potential function is a testing ground for statistical mechanics, for some of the properties of such a system can be obtained analytically by using the theorems of statistical mechanics.

2.5 The Empirical Tight-Binding Approximation

The subject of this thesis is the tight-binding approximation applied to MD simulation in line with [19, 20, 21, 6, 22]. In one sense, this method is in the middle of the two methods for calculating the interatomic forces: ab-initio

calculations and empirical potentials. The empirical tight binding method includes the electronic degrees of freedom, so the quantum mechanical nature of the interatomic interactions are taken into account, so in this respect it is superior to empirical potentials. On the other hand, it is not as rigorous as the ab-initio methods.

The empirical tight-binding method is easy to implement in an MD simulation and the computational cost of calculating the interatomic forces is modest compared to the ab-initio methods, thus the size of the system which can be simulated in a reasonable computer time is much larger.

The tight-binding method can be used as an ab-inito method as well[23], but to permit to the simulation of large systems, we prefer to use the empirical method. Our purpose is to use a scheme which permits the simulation of many particles so that a wide variety of materials phenomena can be investigated with a realistic description of the interatomic interactions, which requires that quantum mechanical effects must be included explicitly.

We develop an MD code which uses the empirical tight binding method to calculate interatomic forces. The results obtained by running this code is in good agreement with the results obtained in [6], which assures us that this formulation is a realistic one. Another advantage of the code is the ease to

modify it in order to simulate systems of different atoms. I will use a similar code to simulate silicon carbide by changing the tight-binding parameters for my Ph.D. thesis.

Chapter 3

Derivation of Empirical Tight-Binding Approximation

3.1 The System

The system I simulated consists of silicon nuclei and electrons. The interactions among these charged particles are, of course, Coulomb interactions. The constituents of nuclei (protons and neutrons) are tightly confined and can be treated as moving as a unit for all practical purposes. The four valence electrons of the silicon atom largely determine the chemical properties of Si.

The Coulomb force acting on a particle with charge q_1 due to another particle located on the origin with charge q_2 can be given as

$$\vec{F} = \frac{1}{4\pi\epsilon_0}\frac{q_1 q_2}{r^3}\vec{r} \tag{3.1}$$

where r is the distance between the two particles and $\vec{r}$ is the radial vector directed towards particle 1, and ϵ_0 is a constant whose value is $\epsilon_0 = 1/(36\pi 10^9)$.

The core electrons constitute a spherical charge cloud between the valence electrons and the Si nucleus. They shield the charge of the nucleus, so the core electrons can be taken into account by assigning a reduced charge to the nucleus.

The two basic types of the particles we are simulating are the cores which consist of the nuclei plus the core electrons and the valence electrons. So we treat the nucleus plus the core electrons as a single unit. This division is our first important approximation.

The electrons have spins of 1/2 so that they are Fermions.[24]The fact that electrons are Fermions has important implications. The Pauli exclusion principle tells us that no two electrons can have exactly the same quantum state in a given system. The statistics of the electrons is of course the Fermi-

Dirac statistics. When two atoms gets close to each other, the electron wavefunctions begin to overlap. Because no two electrons can be in the same quantum state, electrons must rearrange themselves and the energy levels of the electrons will shift. If the rearrangement reduces the total energy of the system, we expect that a bond is formed between the two atoms[25]. If the atoms are forced towards each other, more and more electrons jump to higher energy levels, rapidly increasing the potential energy of the system. The Pauli exclusion principle is responsible for the repulsive force.

3.2 Schrödinger Equation

When we solve the Schrödinger equation, we obtain the total wavefunction of the electron-nuclei system. We know from quantum mechanics that this function contains the maximum amount of the information one can get from a given system.[24] For a many-body system which contains N particles, like ours, the total wavefunction can be written as

$$\Psi = \Psi(x_1, y_1, z_1, ..., x_k, y_k, z_k, ..., x_N, y_N, z_N, t) \tag{3.2}$$

The volume element of the system is

$$dV = dV_1 dV_2 ... dV_k ... dV_N \tag{3.3}$$

$$= dx_1 dy_1 dz_1 ... dx_k dy_k dz_k ... dx_N dy_N dz_N \tag{3.4}$$

Then we can interpret the total wavefunction of the system by writing down the quantity ω

$$\omega(x_1 y_1 z_1 ... x_k y_k z_k ... x_N y_N z_N) = \Psi^* \Psi dV \tag{3.5}$$

where ω is the probability of finding the system in the volume element dV. In quantum mechanics, the physical variables are represented as operators. When we want to know a property of the system, like its momentum, we need to find the expectation value of the relevant operator. The expectation value of an operator is calculated from an inner product which is an integral over the whole space. If we denote the expectation value of an operator by <> and an operator with a $\hat{}$, we can write the expectation value of an operator

$\hat{f}$ as

$$< \hat{f} >= \int \Psi^* \hat{f} \Psi dV \tag{3.6}$$

The time-independent Schrödinger equation can be written as

$$\hat{H}\Psi = E\Psi. \tag{3.7}$$

Clearly, if we substitute 3.7 into 3.6, we obtain the total energy (kinetic+potential) of the system. Here $\hat{H}$ is the Hamiltonian operator of the system which corresponds to the total energy of a classical system. The Hamiltonian of our system can be written as the sum of the kinetic energies of the electrons and the nuclei plus their Coulomb interaction energy. We can write $\hat{H}$ as

$$H = H_{el} + H_{ion} + H_{el-ion} + H_{ex} \tag{3.8}$$

We do not have the last term in our calculations which represents an external potential (like an applied electric field). For the electron part we can write

$$H_{el} = H_{el,kin} + H_{el-el} = \sum_k \frac{p_K^2}{2m} + \frac{1}{8\pi\epsilon_0} \sum_{kk'}{}' \frac{e^2}{|\vec{r}_k - \vec{r}_{k'}|} \tag{3.9}$$

The interaction is naturally the Coulomb interaction. The sums are over all electron indices except $k = k'$ for the interaction term. $\vec{p}_k, \vec{r}_k$, and m are the momentum, position and mass of an electron of index k. For the ion part of the Hamiltonian we can write

$$H_{ion} = H_{ion,kin} + H_{ion-ion} = \sum_i \frac{P_i^2}{2M_i} + \frac{1}{2} \sum_{ii'}{}' V_{ion}(\vec{R}_i - \vec{R}_{i'}) \tag{3.10}$$

Ion parameters are shown with capital letters. For the electron-ion interaction we put

$$H_{el-ion} = \sum_{k,i} V_{el-ion}(\vec{r}_k - \vec{R}_i). \tag{3.11}$$

The solution of 3.7 is extremely difficult except for very simple systems, like the H atom. Even for the He atom,[24] we need to make approximations in order to solve 3.7.

3.3 Adiabatic Approximation

Now I will make an approximation to the total wavefunction in order to greatly simplify the solution to the 3.7. The basis of this approximation is that because electrons have a much smaller mass than the nuclei, they move much faster than the nuclei, so electronic system adjusts itself almost instanteneously to the changes in the configuration of the nuclei. This fact enables us to decouple the electronic and the ionic motion. I will desribe this approximation briefly, a more rigorous discussion can be found in [26].

Let us assume that we can write a Schrödinger equation for the electrons as

$$(\hat{H}_{el} + \hat{H}_{el-ion})\psi = E_{el}\psi \tag{3.12}$$

and that 3.2 can be approximated as

$$\Psi = \psi(\vec{r}_1 ... \vec{r}_N; \vec{R}_1 ... v R_{N'})\phi(\vec{R}_1 ... \vec{R}_{N'}) \tag{3.13}$$

where the ψ are solutions of 3.12 and N, N' denote the number of electrons and ions, respectively. By substituting 3.13 into 3.7 and remembering that

$\nabla_i^2\psi\phi = \psi\nabla_i^2\phi + \phi\nabla_i^2\psi + \nabla_i\psi.\nabla_i\phi$, we obtain the following equation:

$$\hat{H}\Psi = (H_{el} + H_{ion} + H_{el-ion})\psi\phi \tag{3.14}$$

$$= \psi(H_{ion} + E_{el})\phi - \sum_i \frac{\hbar^2}{2M_i}(\phi\nabla_i^2\psi + 2\nabla_i\phi.\nabla_i\psi). \tag{3.15}$$

The last term prevents us from decoupling the electronic and ionic motion, because it depends on the both electronic and ionic coordinates in an inseparable way. If we neglect this term, for the ionic motion we would have

$$(H_{ion} + E_{el})\phi = E\phi \tag{3.16}$$

We can make this approximation because of the large difference between electronic and ionic velocities. So what we need to do is to solve 3.12 by assuming that the cores are fixed and then substitute the electronic energy into 3.16 and solve it for the ionic motion. In the application of the empirical tight binding method in the MD simulation, the electronic energy is calculated first and the ionic trajectory is calculated by using this ionic energy.

3.4 One-Electron Approximation

The basis of the tight-binding method is the one-electron approximation, which decouples the motion of each electron from the rest of electrons in a way which includes the many-body effects of the rest of particles. I again give only a brief description of this approximation. A precise argument of this approximation can be found in [27].

As a first approximation, let us assume that the electronic wavefunction of 3.12 can be written as

$$\psi(\vec{r}_1...\vec{r}_N) = \varphi_1(\vec{r}_1)\varphi_2(\vec{r}_2)..\varphi_N(\vec{r}_N) \tag{3.17}$$

What we need to find is a set of φ's which minimize the ground state energy of 3.12. The variational approach is the key to the solution of this problem. The result is that

$$\begin{aligned} &[-\frac{\hbar}{2m}\nabla^2 + V(\vec{r}) + \frac{e^2}{4\pi\epsilon_0}\sum_{k(\neq j)}\int\frac{|\varphi_k(\vec{r}')|^2}{|\vec{r}-\vec{r}'|}d\tau']\varphi_j(\vec{r}) \\ = \;& E_j\varphi_j(vr). \end{aligned} \tag{3.18}$$

It turns out that in order to be more precise, we need to use a linear com-

bination of multiplications of trial wavefunctions like 3.17, because electrons obey the Pauli exclusion principle, so that the electronic wavefunction must be zero if the two electron coordinates equal to each other(for example, $\psi(\vec{r}_1, \vec{r}_2 = \vec{r}_1, \vec{r}_3, ..., \vec{r}_N) = 0$. Also the electronic wavefunction must be antisymmetric under the exchange of two electronic coordinates. These two conditions are satisfied by the Slater determinant

$$\psi = (N!)^{-1/2} \begin{vmatrix} \varphi_1(\vec{q}_1) & \cdots & \varphi_N(\vec{q}_1) \\ \vdots & & \vdots \\ \varphi_1(\vec{q}_N) & \cdots & \varphi_N(\vec{q}_N) \end{vmatrix} \tag{3.19}$$

where the factor in front of the determinant is added for normalization purposes. This form satisfies the Pauli principle. If two electrons are interchanged, two columns of the determinant are interchanged and ϕ changes sign. Again, if the two electrons have the same coordinates, two columns are identical and ϕ vanishes.

Now again we need substitute 3.19 into 3.12 and apply the variational principle to find the best trial functions which minimize the ground state energy of the electronic system. Again, I will only give the result. The

derivation of the result can be found in [27].

$$[-\frac{\hbar^2}{2m}\nabla^2 + V(\vec{r})]\varphi_j(\vec{r}) + \frac{e^2}{4\pi\epsilon_0}\sum_{k(\neq j)}\int \frac{|\varphi_k(\vec{r}')|^2}{|\vec{r}-\vec{r}'|}d\tau'\varphi_j(\vec{r}) \quad (3.20)$$
$$- \frac{e^2}{4\pi\epsilon_0}\sum_{\substack{k(\neq j)\\ spin||}}\int \frac{\varphi_k^*(\vec{r}')\varphi_j(\vec{r}')}{|\vec{r}-\vec{r}'|}d\tau'\varphi_k(\vec{r}) = E_j\varphi_j(\vec{r})$$

where

$$\overline{\rho}_j^{HF} = -e\sum_k \frac{\varphi_k^*(\vec{r}')\varphi_j(\vec{r}')\varphi_j^*(\vec{r})\varphi_k(\vec{r})}{\varphi_j^*(\vec{r})\varphi_j(\vec{r})} \quad (3.21)$$

and

$$\rho(\vec{r}') = \sum_j -e|\varphi_j(\vec{r})|^2. \quad (3.22)$$

Even though we have simplified 3.12 greatly by the one-electron approximation, the solution of 3.20 is still extremely difficult and must be done self-consitently. We need to make further assumptions to solve 3.20.

3.5 The Linear Combination of Atomic Orbitals

The basis of the one-electron approximation was discussed above. Now I will make further approximations to reach a simpler equation which is used in the code I wrote. Our ionic Hamiltonian 3.10 contains contributions from the electronic energy and also from core-core interactions. I will assume that the electronic wavefunction can be approximated as a linear combination of atomic orbitals. I used only four atomic orbitals, one s and three p functions. The form of these functions can be found in [2].

$$\psi = \sum_{m,n} \lambda_{mn} f_m(\vec{r} - \tau_n) \tag{3.23}$$

Now let us try to find the forces acting on ion cores due to the electronic structure. To find the electronic force acting on core n, we need to use the Hellman-Feynman theorem[28]:

$$\vec{F}_n = -\sum_j < \psi_j | \partial \hat{H}_{el} / \partial \vec{\tau}_n | \psi_j > = -\sum \frac{\partial}{\partial \vec{\tau}_n} < \psi_j | \hat{H}_{el} | \psi_j > \tag{3.24}$$

where the sum is over the occupied states. By substituting 3.23 into 3.24, we obtain that

$$\vec{F}_n = - \sum_{j,m,m',n'} \lambda^*_{jmn} \lambda_{jm'n'} \partial H_{m'n'mn} / \partial \vec{\tau}_n \tag{3.25}$$

where

$$H_{m'n'mn} = < f_{m'}(\vec{r} - \vec{\tau}_n) | \hat{H}_{el} | f_m(\vec{r} - \vec{\tau}_n > \tag{3.26}$$

The calculation of these matrix elements is beyond the scope of my masters thesis. A good description of the calculation of these matrix elements can be found in [23]. If the explicit calculation of these matrix elements is used to calculate the electronic forces acting on cores, this method would be an ab-initio method. Instead, I will use these matrix elements in parametrized form. Let us use the nearest-neighbor approximation. In the case of the diamond cubic Si, we have a total of four nearest neighbors per Si atom. Harrison showed that[25] the matrix elements for the nearest-neighbor approximation

has $\frac{1}{r^2}$ dependence. The matrix elements can be written as[2]

$$
\begin{aligned}
H_{ss} &= V_{ss\sigma} \\
H_{sx} &= -H_{xs} = cos\theta_x V_{sp\sigma} \\
H_{xx} &= cos^2\theta_x V_{pp\sigma} + sin^2\theta_x V_{pp\pi} \\
H_{xy} &= H_{yx} = cos\theta_x cos\theta_y (V_{pp\sigma} - V_{pp\pi})
\end{aligned}
\tag{3.27}
$$

where θ_x is the angle between the vector connecting the two nearest neighbor and the x axis. From Harrison's work [25] we know that

$$V_{abc} = V_{abc}\frac{d_0^2}{r^2} \tag{3.28}$$

where V_{abc} is a constant and d_0 is the equilibrium nearest-neighbor distance of Si. The V_{abc}'s are tabulated for several covalent materials[25], so they can be inserted into our equations directly.

3.6 Construction of Repulsive Pair Potential

As we know, the ionic Hamiltonian 3.10 contains a repulsive term which is represented as a pair potential function. We can think the origin of this potential as the coulomb repulsion of the positive-charged cores. But the shielding by electrons changes the shape of this potential from a purely coulombic potential. A good way to obtain this potential is to subtract the attractive potential due to electronic system from the total potential energy of the system per silicon atom. The total energy of the silicon per atom can be obtained from the ab-initio calculations of silicon. Such a calculation was carried out by Yin and Cohen [29] and I fit the total energy per silicon to a curve

$$E_{tot}(r) = E_0[1 + (r - r_0)/A]exp[-(r - r_0)/A] \tag{3.29}$$

Then the repulsive pair potential can be written as

$$\phi(r_{ij}) = \frac{1}{2}[E_{tot}(r_{ij}) - E_{BS}(r_{ij})] \tag{3.30}$$

where E_{BS} is the sum of the eigenvalues of the Hamiltonian matrix for the occupied part of the band structure divided by the number of atoms. The occupied part of the band structure consists of lowest $2N$ eigenvalues of the Hamiltonian matrix.

3.7 Tight-Binding Parameters

I obtained the tight-binding parameters from Chadi's work on Si surfaces[19]. They are given as

$$
\begin{aligned}
V_{ss\sigma}(d_0) &= -1.94eV \\
V_{sp\sigma}(d_0) &= 1.75eV \\
V_{pp\sigma}(d_0) &= 3.05eV \\
V_{pp\pi}(d_0) &= -1.08eV; E_s = -5.20eV \\
E_p &= 1.2eV
\end{aligned}
\tag{3.31}
$$

The fit to the total potential function produced the following parameters:

$$E_0 = -4.8060eV$$

$$r_0 = 2.3627A$$

$$A = 0.5076A \tag{3.32}$$

Chapter 4

Molecular Dynamics Simulation

4.1 The Code

I wrote code QUNTUM to simulate silicon atoms by using the empirical tight-binding method I have described. The flowchart of the program is shown in fig.4.1.

I used a total of 8 atoms in my simulations. The total energy of silicon per atom in diamond structure as a function of the nearest-neighbor distance is taken from [29]. The curve fit to the data is shown in Fig.4.2.

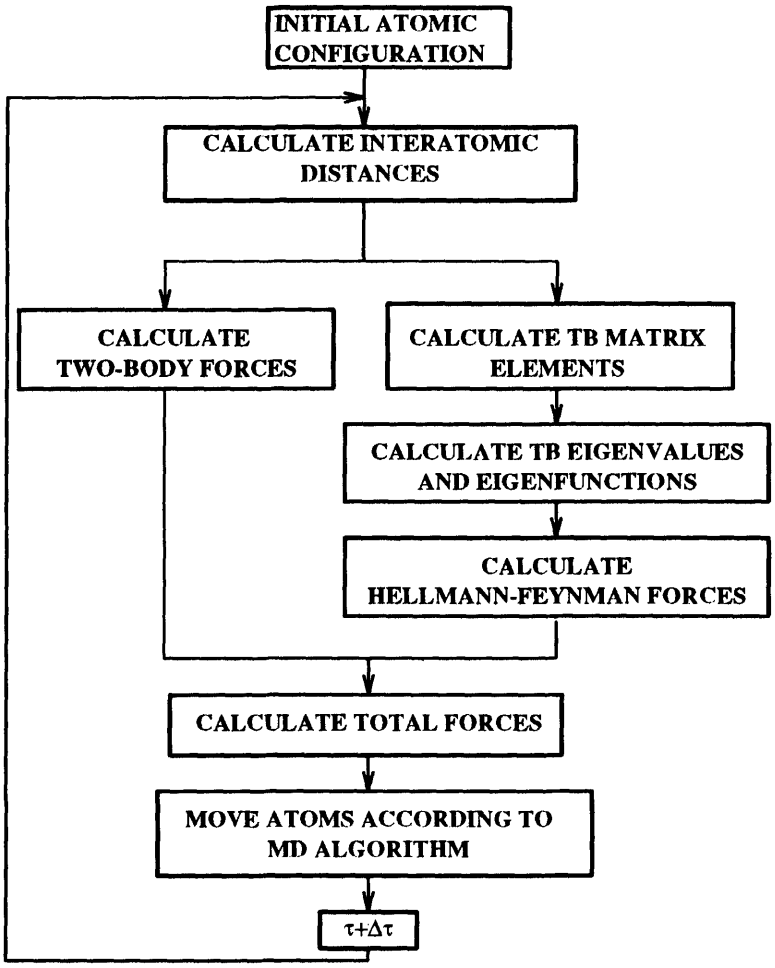

Figure 4.1: Flowchart of code QUANTUM.

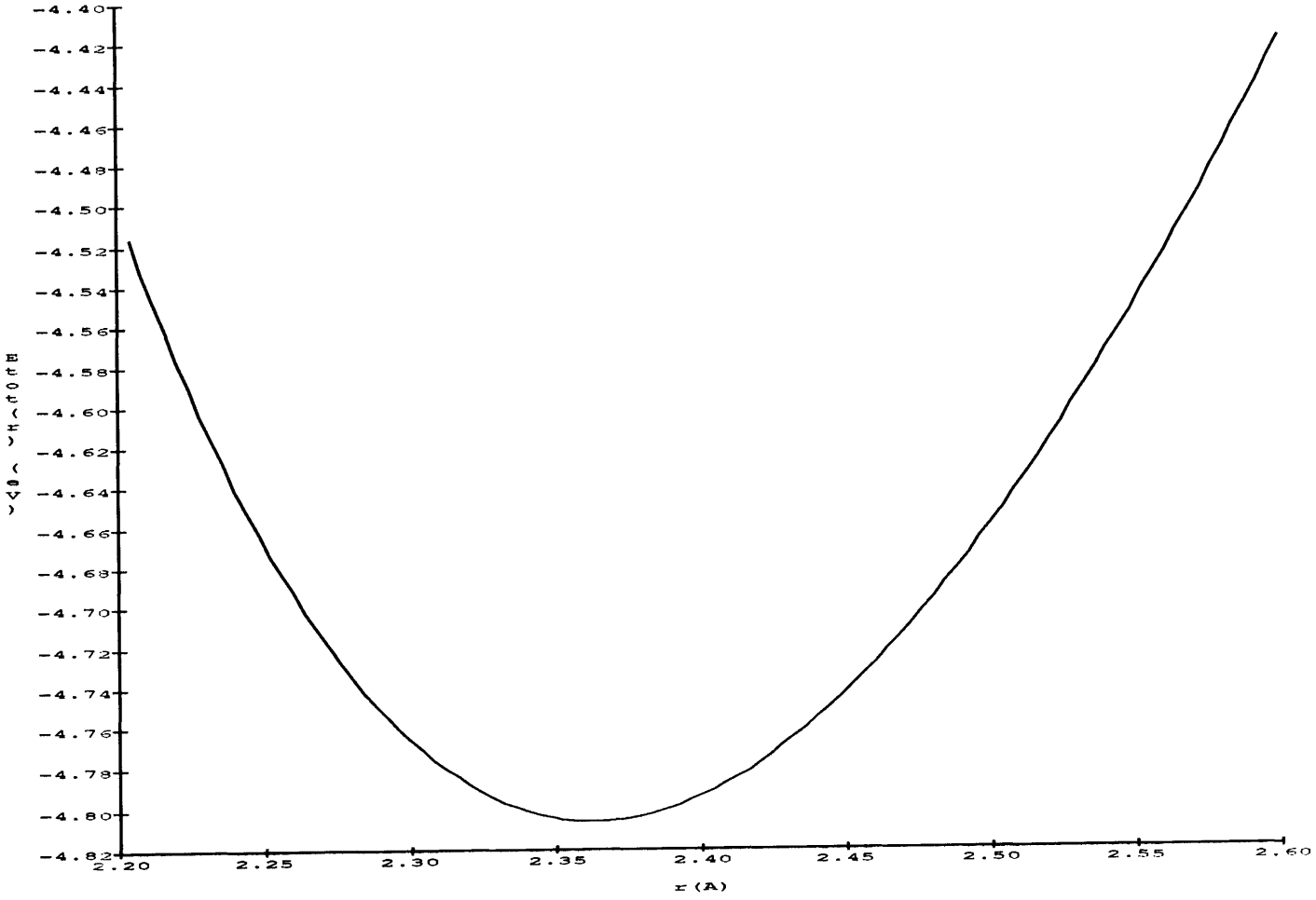

Figure 4.2: Total potential energy of Si in diamond structure as a function of the nearest-neighbor distance.

Clearly, the equilibrium nearest neighbor distance is the distance where the derivative of the total energy with respect to the nearest neighbor is zero. From fig.4.2 we see that the equilibrium nearest-neighbor distance for silicon in the diamond structure is $r_0 = 2.37A$.

To obtain the band-structure energy of silicon per atom, I ran my code for $T = 0K$ for various nearest-neighbor distances and calculated the band-structure energy. Then I plotted the band structure energy as a function of the nearest-neighbor distance (Fig.4.3)and fit a polynomial to this curve.

The band structure energy calculated by code QUANTUM is in the proximity of a couple of percent to the calculations carried out in [6] and shown in Fig. 4.4.

Then we found the repulsive pair-potential by subtracting the band-structure energy from the total energy and plotted it (Fig.4.5) and fit it to a third-order polynomial.

After finding the repulsive potential, I plugged it in the MD code and run it for 6 different temperatures from $100K$ to $600K$ for 2000 time-steps after 500 equilibration steps and then stored the coordinates of each atom. The time step for my simulation was $5.472x10^{-16}s$. I found the various physical properties by using the phase trajectories produced by code QUANTUM and

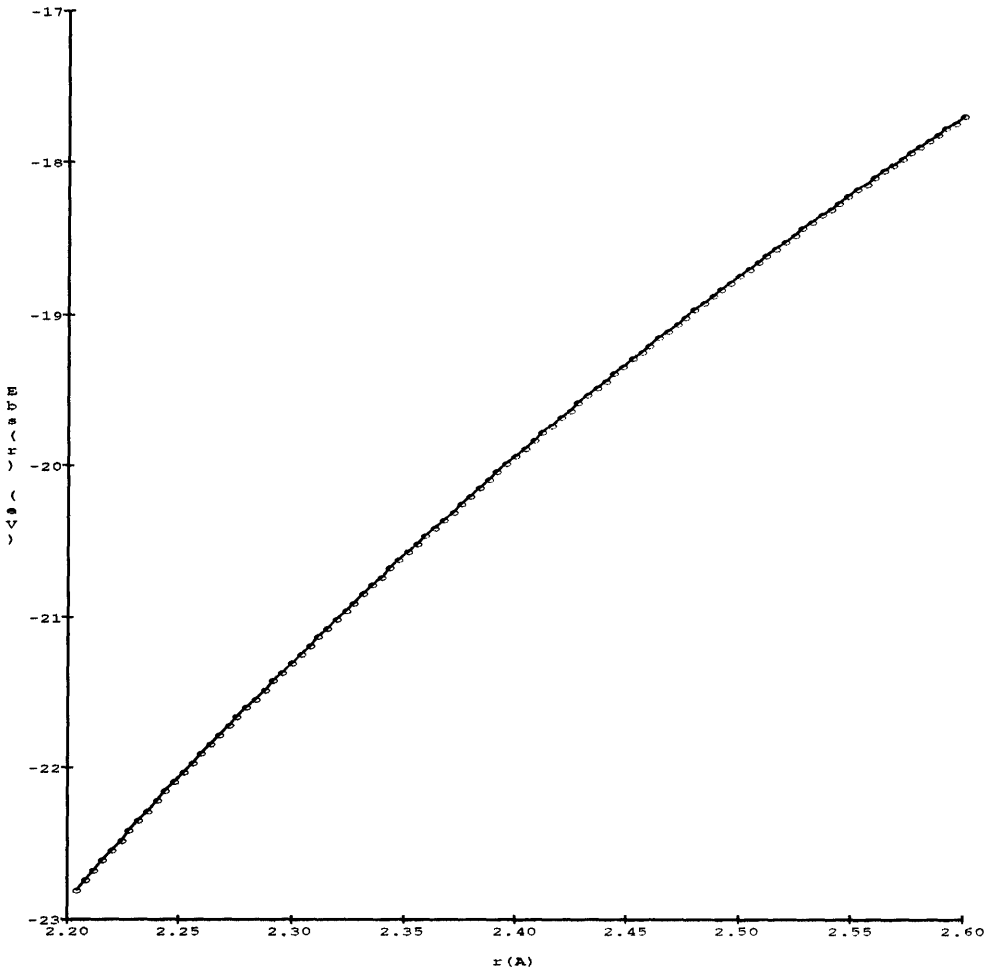

Figure 4.3: Band-structure energy per atom of silicon in diamond structure calculated by code QUANTUM.

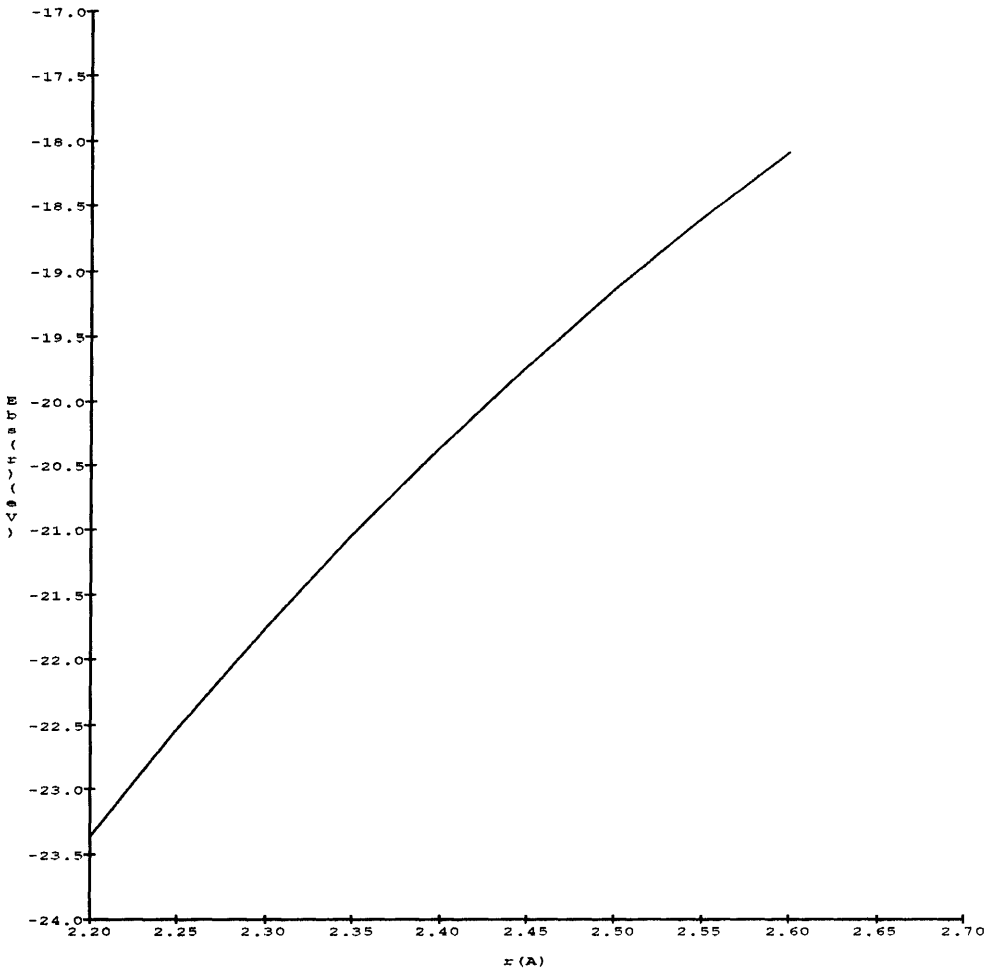

Figure 4.4: The band-structure energy per silicon atom in the diamond-cubic structure taken from literature.

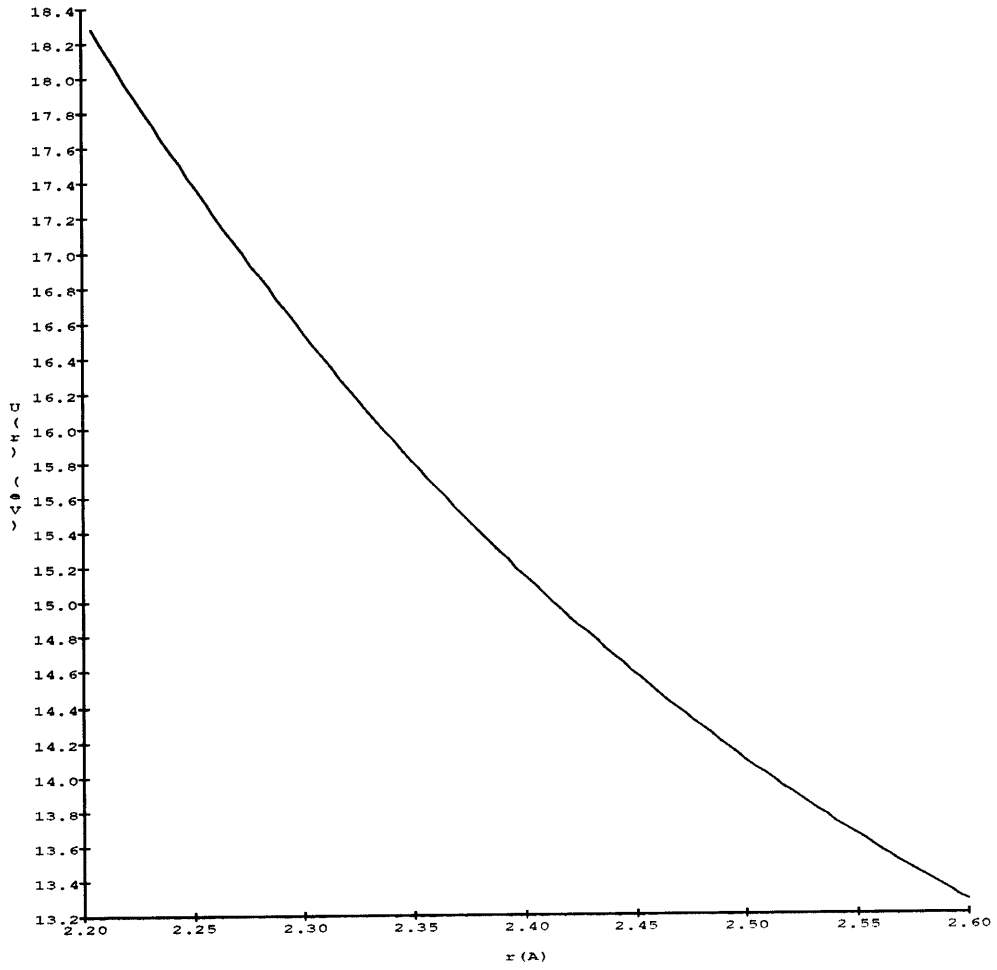

Figure 4.5: The repulsive pair potential per silicon atom in diamond structure.

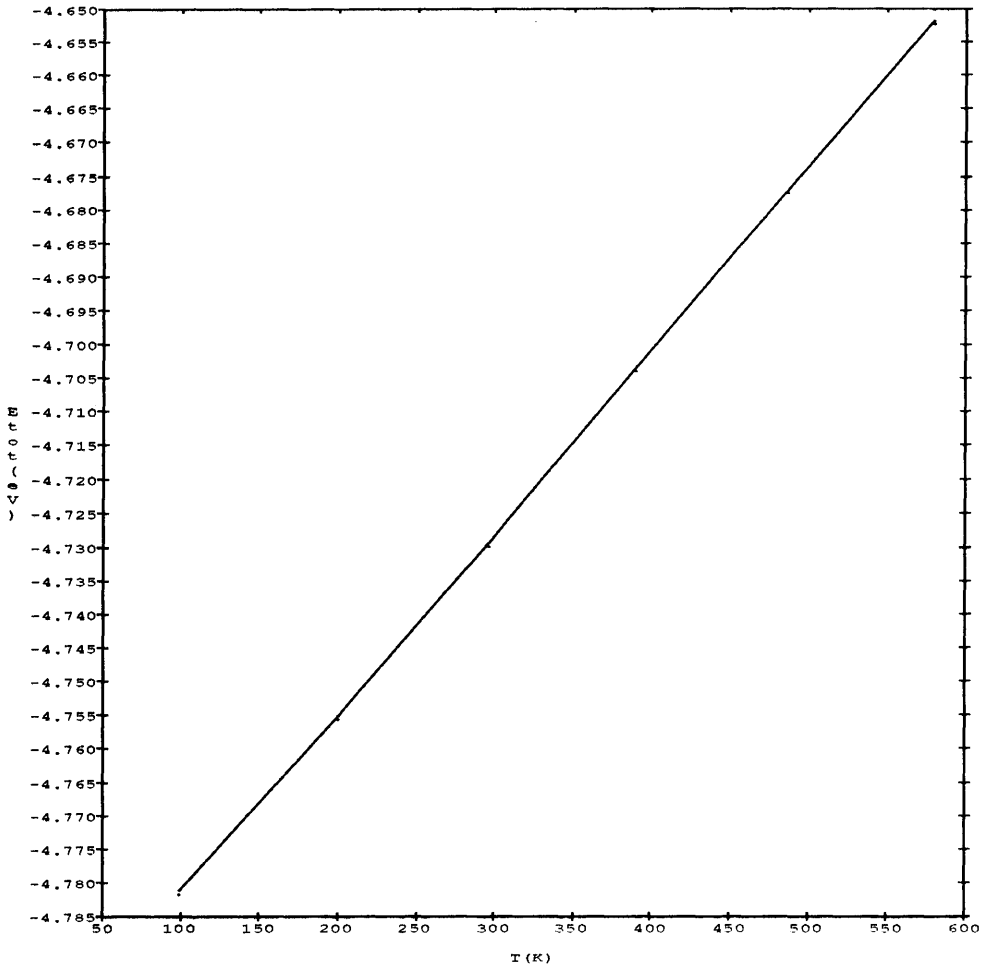

Figure 4.6: The total energy of Si per atom as a function of temperature for constant volume case.

then stored in a file.

4.2 Results and Discussions

I obtained the total energy of the system as a function of temperature under constant volume(Fig.4.6). The result is very close to the one in [6] as shown in Fig 4.7. I am adding a copy of their results at the end of my thesis.

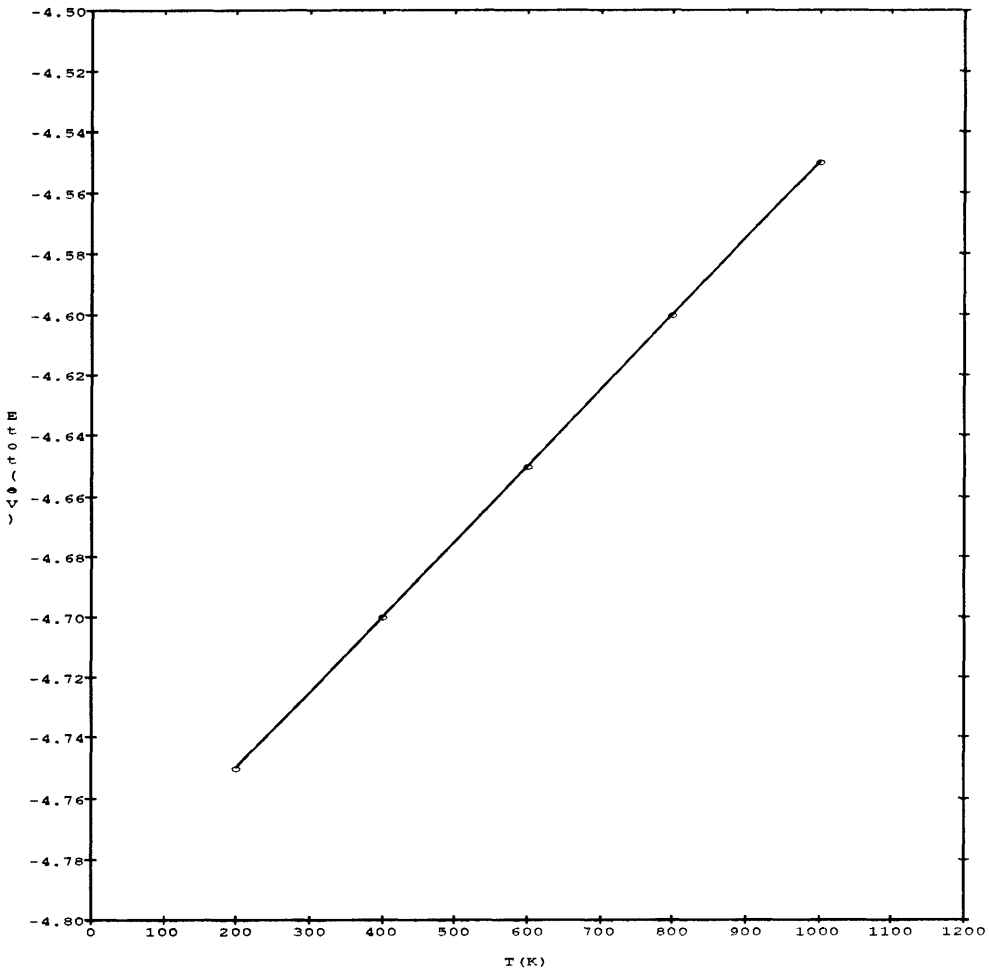

Figure 4.7: The total energy of Si per atom as a function of temperature for constant volume case taken from literature.

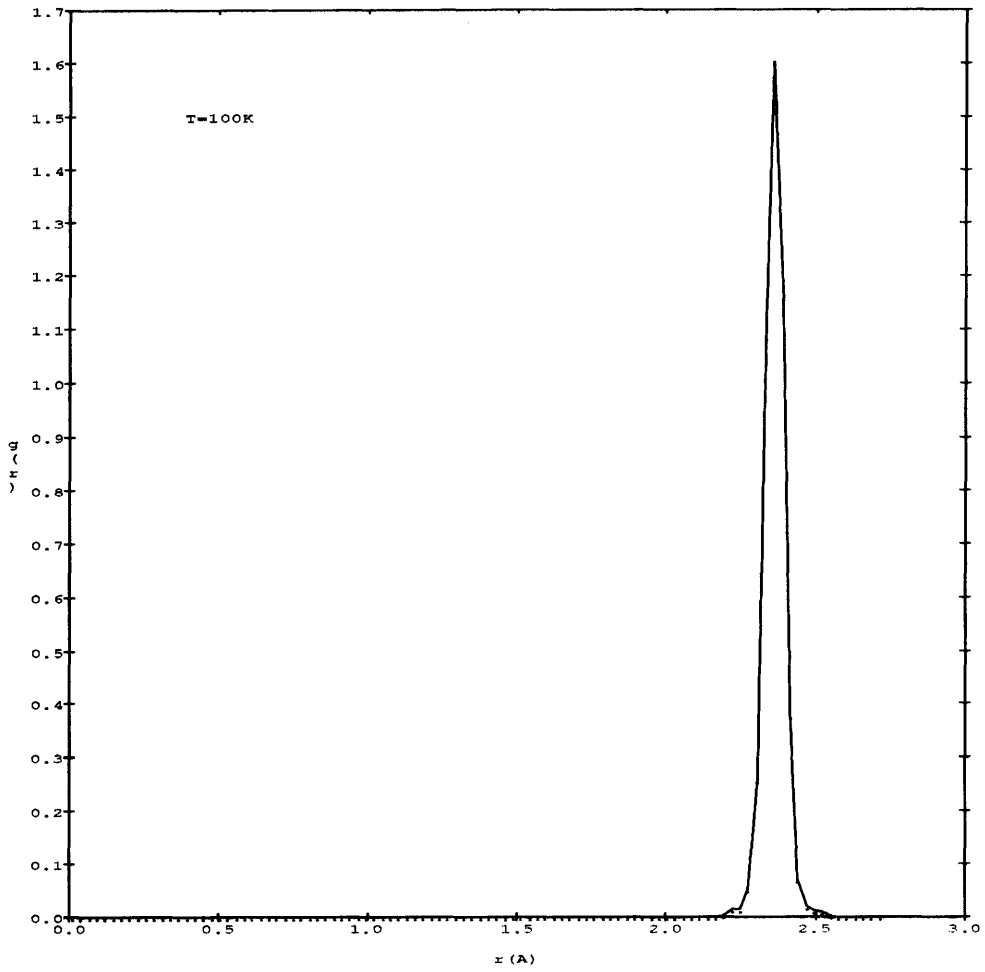

Figure 4.8: g(r) for T=100K

Afterwards, I obtained the radial distribution function of the system for each temperature. Because of the small system size, only the peak corresponding to the first nearest neighbor can be plotted. but these plots show that the diamond structure is stabilized by the empirical tight-binding method. Also by viewing the evolution of the system in time by using the graphics computer SiliconGraphics Personal Iris, the stability of the system was confirmed.

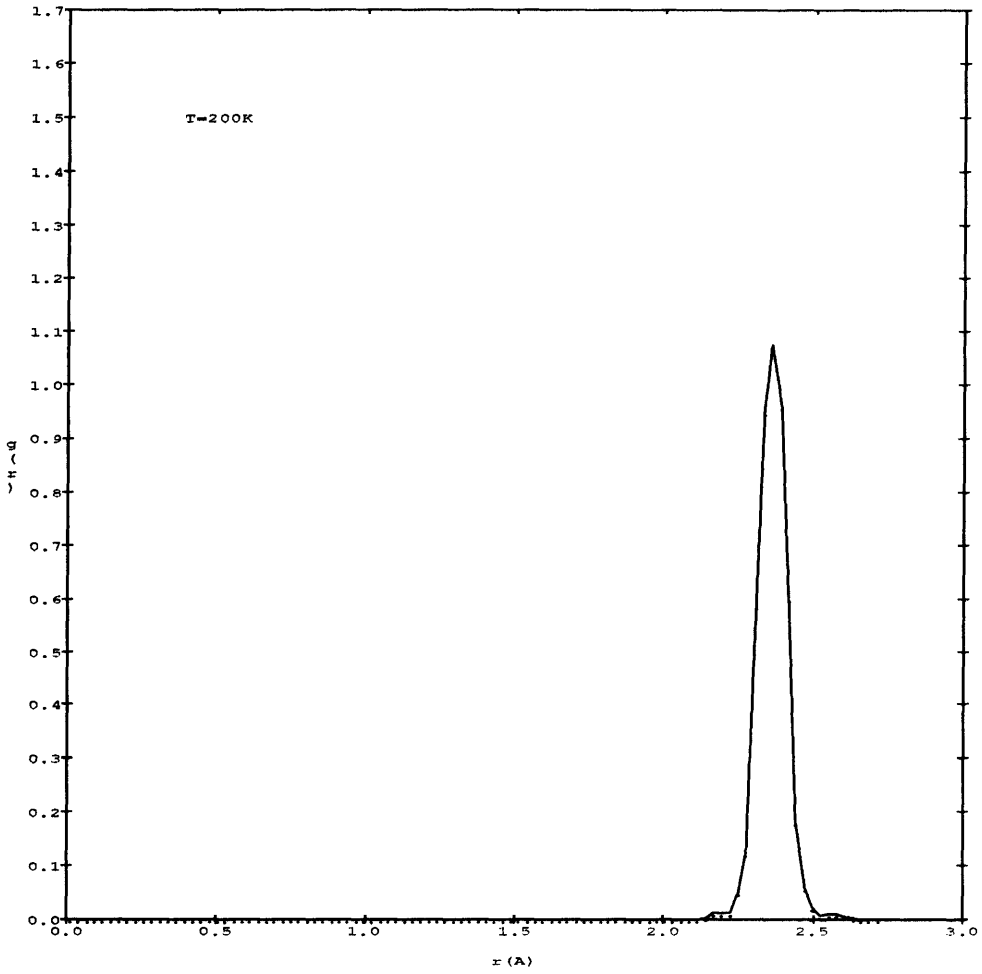

Figure 4.9: g(r) for T=200K

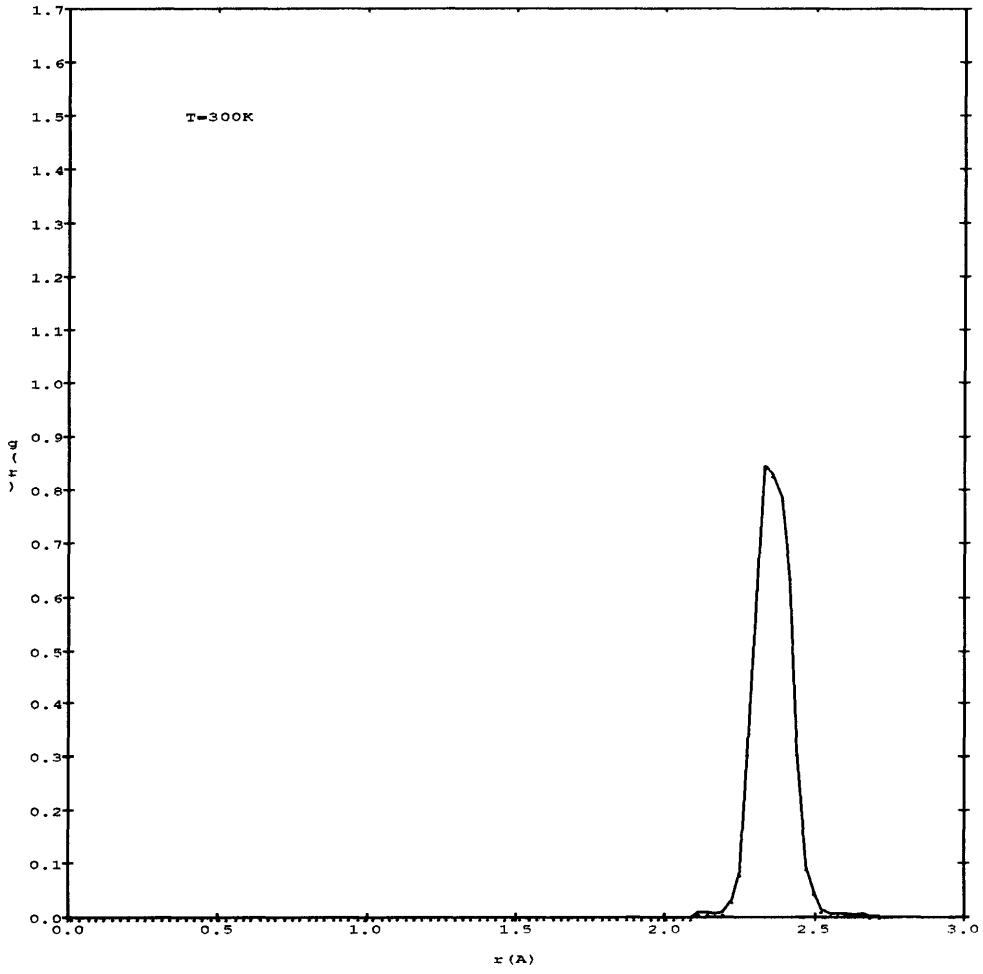

Figure 4.10: g(r) for T=300K

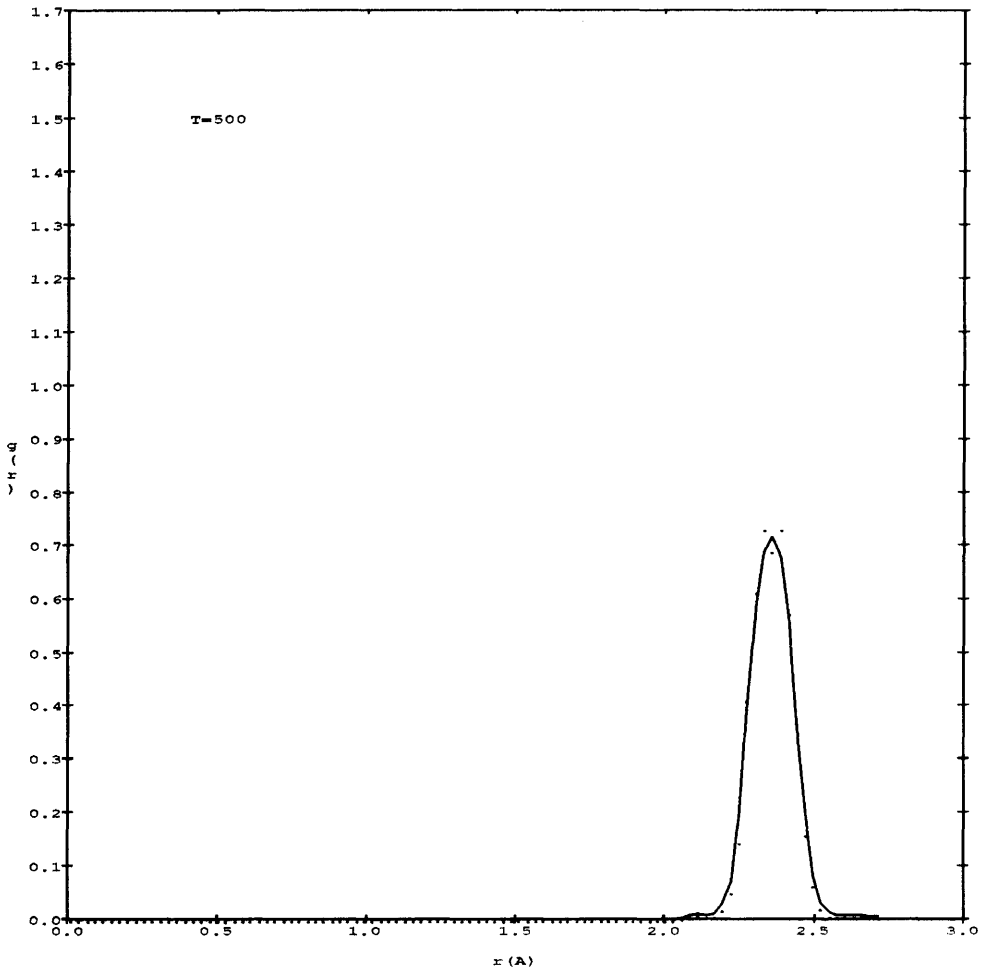

Figure 4.11: g(r) for T=500K

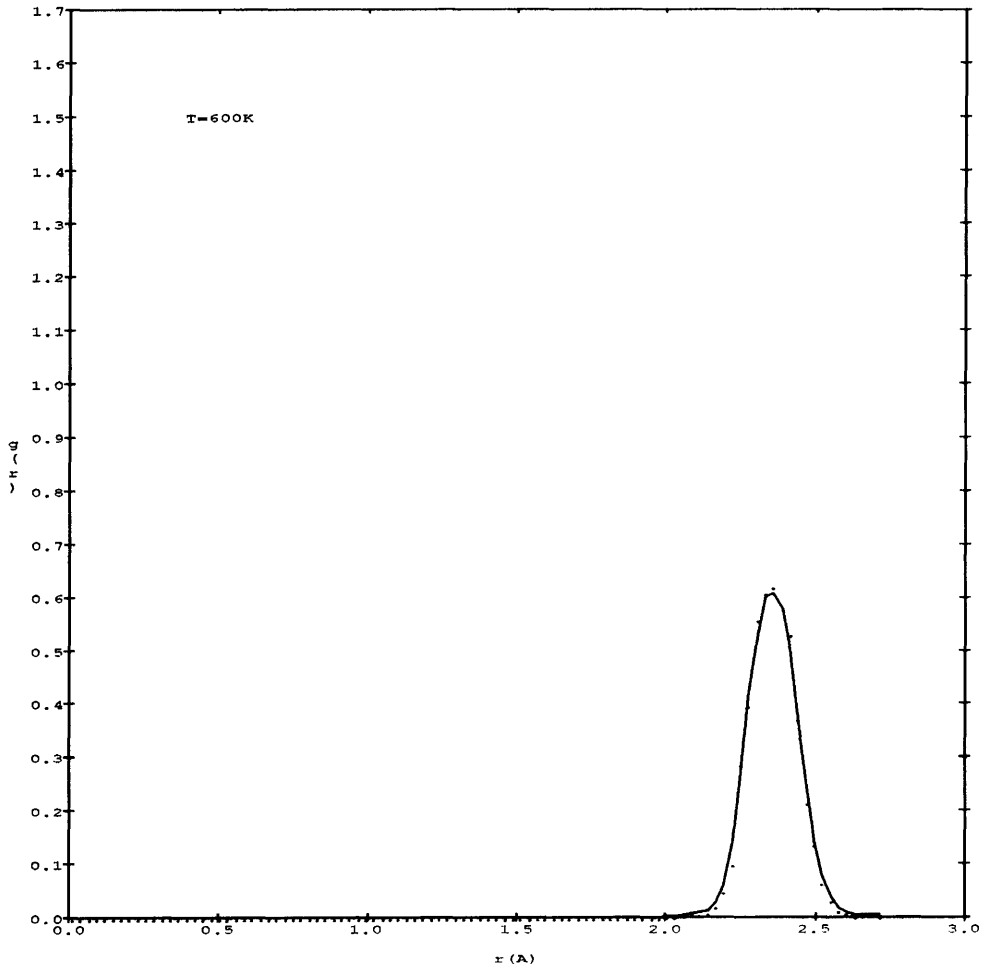

Figure 4.12: g(r) for T=600K

4.3 Conclusions

Clearly, the empirical tight-binding approximation provides us with a practical and powerful method for MD simulation of atomic systems. The method is practical, because the Hellman-Feynman forces are easy to calculate, and the computational cost of the simulation is moderate compared to the ab-initio methods, so that we are able to simulate up to 1000 atoms in a given system. Also by changing only a couple of tight-binding parameters which can be obtained from literature easily, we are able to simulate a large variety of systems containing different kinds of atoms like carbon, silcon, nitrogen...

The empirical tight-binding method is also powerful, because we include the electronic degrees of freedom explicitly in the calcualtions, so the quantum mechanical nature of forces is taken into account as well. Even though the calculation of interatomic forces is not as rigorous as it is in the case of ab-initio methods, the reduced computational load enables us to simulate up to 1000 particles compared to up to 100 in case of ab-initio methods.

I am planning to use the set of codes I developed for silicon to simulate SiC and possibly carbon by using appropriate tight-binding parameters for each system. Even though there are empirical potential functions devel-

oped for these materials, they are unable to describe the physical properties of those systems as accurately as we want. I predict that the empirical tight-binding approximation will provide us with a deeper understanding of materials phenmena in those materials than it was previously possible.

Bibliography

[1] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids.* Clarendon Press, 1987.

[2] J.C. Slater and G.F. Koster. *Phys. Rev.*, 94:1498, 1954.

[3] F.H. Stillinger and T.A. Weber. *Phys. Rev.*, B(31):5262, 1985.

[4] T.A. Weber and E. Helfand. *J. Phys. Chem.*, 87:2881, 1983.

[5] F.H. Stillinger and T.A. Weber. *Phys. Rev. A*, 28:2408, 1983.

[6] C.Z. Wang, C.T. Chan, and K.M. Ho. *Phys. Rev. B*, 39:8586, 1988.

[7] P. Hohenberg and V. Kohn. *Phys. Rev.*, 136:B864, 1964.

[8] W. Kohn and L.J. Sham. *Phys. Rev.*, 140:A1133, 1965.

[9] R. Car and M. Parrinello. *Phys. Rev. Lett.*, 55:2471, 1985.

[10] R. Car and M. Parrinello. *Phys. Rev. Lett.*, 62:555, 1989.

[11] R. Car and M. Parrinello. *Phys. Rev. Lett.*, 63:988, 1989.

[12] I. Stich, R. Car, and M. Parrinello. *Phys. Rev. Lett.*, 63:2240, 1989.

[13] S. Lundqvist and N.H. March. *Theory of Inhomogeneous Electron Gas.* Plenum, 1983.

[14] I. Stich, R. Car, and M. Parrinello. *Molecular Physics*, 70:921, 1989.

[15] L. Verlet. *Phys. Rev.*, 159:98, 1967.

[16] S. Daw and M.I. Baskes. *Phys. Rev. B*, 29:6443, 1984.

[17] S. Daw and M.I. Baskes. *Phys. Rev. Lett.*, 50:1285, 1983.

[18] A.E. Carlsson. *Derivation of Angular Forces for Semiconductors and Transition Metals.* Springer-Verlag, 1990.

[19] D.J. Chadi. *Phys. Rev. Lett.*, 41:1062, 1978.

[20] D.J. Chadi. *Phys. Rev. B*, 29:785, 1984.

[21] D.J. Chadi and J.R. Chelikiwsky. *Phys. Rev. B*, 24:4892, 1981.

[22] C.Z. Wang, C.T. Chan, and K.M. Ho. *Phys. Rev. B*, 40:3390, 1989.

[23] H. Eschrig. *Optimized LCAO Method.* Springer-Verlag, 1989.

[24] W. Greiner. *Quantum Mechanics, An Introduction.* Springer-Verlag, 1989.

[25] W.A. Harrison. *Electronic Structure and the Properties of Solids.* Freeman, 1980.

[26] A.Haugh. *Theoretical Solid State Physics.* Pergamon Press, 1972.

[27] O. Madelung. *Introduction to Solid State Theory.* Springer-Verlag, 1981.

[28] A. Feynman. *Phys. Rev.*, 60:2240, 1942.

[29] M.T. Yin and M.L. Cohen. *Phys. Rev. B*, 26:3259, 1982.

CODE QUANTUM

```
/*

This program carries out the molecular dynamics simulation of silicon
in the diamond-cubic structure by using the empirical tight-binding
approximation to calculate interatomic forces. Number of
particles, system dimensions, number of time steps
and the length of each time-step are given in the input file.

Written by:
  Ahmet Isik, Massachusetts Institute of Technology,                    10
  Cambridge, MA 1991.

*/
float Vssm=(-1.94),Vspm=1.75, Vppm=3.05, Vppi=(-1.08);
```

```
float Es=(-5.20), Ep=1.2;
float A,A0=(-23.37), A1=17.32, A2=(-12.42),A3=5.25, AA=0.5076;
float d02=5.5823,E0=(-4.806), rb=2.2, r0=2.3627,SN;
float side[3];
float inter5[4];
int N,nn[3],nu[3];                                                    20
#define NA 9
#define NB  33
#include<stdio.h>
#include<math.h>
#define euler(a, b, c, d, e, f) (pow(a-b, 2.)+pow(c-d, 2.)
        +pow(e-f, 2.))
#define judy(a, b) euler(y[0][0][a],y[0][0][b],y[0][1][a],
 y[0][1][b],y[0][2][a],y[0][2][b])
float ext(),y[2][3][NA],dy[2][3][NA],side[3];
float mat[4],h[NB][NB],ter[4];                                        30
main()
{
 FILE *fpi,*fpo;
 int ni;
 float TEMP;
 float EPS, DT,hmin,x1=0,x2=100.;
```

```
fpi=fopen("input", "r");
fpo=fopen("output","w");
mat[0]=Vssm*d02;
mat[1]=Vspm*d02;
mat[2]=Vppm*d02;
mat[3]=Vppi*d02;

inter5[0]=21.565502;
inter5[1]=(-219.828989);
inter5[2]=746.270717;
inter5[3]=(-843.663299);

ter[0]=363.638069;
ter[1]=(-370.624998);
ter[2]=132.386506;
ter[3]=(-16.026109);

srandom(0.8);
fscanf(fpi, "%f %f %f %f %f %d %d %d %d",
  &TEMP,&DT,&hmin,&A,&EPS,nu,nu+1, nu+2, &ni);
TEMP=100.;
DT=0.005;
```

```
 A=2.3627;
 nu[0]=1;nu[1]=1;nu[2]=1;                                                    60
 N=8*nu[0]*nu[1]*nu[2];
 printf("\n\n\n\nnumber of atoms.....:%d\n\n", N);
 printf("nearest-neighbor distance.....:%f\n\n", A);
 printf("temperature(K).....:%f\n\n", TEMP);
 printf("time step for integration.....:%f\n\n", DT);
 printf("maximum permissible error.....:%f\n\n", EPS);
 elmas();
 maxwell(TEMP);
 simulate(DT,fpo);
}                                                                            70
/*This function produces the diamond structure.*/
elmas()
{
 int a=0,i,j,k,l,p,z,x,yy,f,f1;
 float add,c[3],AU=8,d,d2,d4,cy,cz,cu[3][10];
 d=2.3094*A;
 for(i=0;i<=2;i++){
  side[i]=d*nu[i];
 }
 d2=d/2.;                                                                    80
```

```
d4=d/4.;
/*Calculation of the unit cube*/
cu[2][1]=cu[2][2]=0.; cu[2][3]=cu[2][4]=d4;
cu[2][5]=cu[2][6]=d2; cu[2][7]=cu[2][8]=d2+d4;
cu[1][1]=cu[1][5]=0.; cu[1][2]=cu[1][6]=d2;
cu[1][3]=cu[1][7]=d4; cu[1][4]=cu[1][8]=d2+d4;
cu[0][1]=cu[0][6]=0.; cu[0][2]=cu[0][5]=d2;
cu[0][3]=cu[0][8]=d4; cu[0][4]=cu[0][7]=d2+d4;
for(i=1; i<=nu[2]; i++){
 c[2]=(i-1)*d;                                                    90
 for(j=1; j<=nu[1]; j++){
  c[1]=(j-1)*d;
  for(k=1; k<=nu[0]; k++){
   c[0]=(k-1)*d;
   for(p=1; p<=AU; p++){
    a++;
    for(l=0; l<=2; l++){
     y[0][l][a]=cu[l][p]+c[l];
    }
   }                                                              100
  }
 }
```

```
 }
}
maxwell(TEMP)
 float TEMP;
{
 int i,j,si,n;
 float r1,r2,vel(),sum[3];
 for(i=0; i<=2; i++){                                          110
  sum[i]=0.;
  srandom(0.5);
  for(j=1; j<=N; j++){
   r1=random()/(pow(2.,31.)-1.);
   r2=random()/(pow(2.,31.)-1.);
   if(r2<.5)
    si=(-1);
   else
    si=1;
   y[1][i][j]=si*vel(TEMP, r1);                                120
   sum[i]+=y[1][i][j];
  }
 }
 for(i=0; i<=2; i++){
```

```
 sum[i]/=N;
 for(j=1; j<=N; j++){
  y[1][i][j]-=sum[i];
 }
}
r1=0.;                                                          130
sum[0]=sum[1]=sum[2]=0.;
for(i=1;i<=N;i++){
 for(j=0;j<=2;j++){
  r1+=y[1][j][i]*y[1][j][i];
 }
}
r1=r1*.5/N;
r1=sqrt(1.5*8.614e-5*TEMP/r1);
for(i=1;i<=N;i++){
 for(j=0;j<=2;j++){                                             140
  y[1][j][i]*=r1;
 }
}
}
float vel(TEMP,r)
 float TEMP, r;
```

```
{
 int i=0;
 float g,g1,u=1.e-5, p, K=8.614e-5,erf();
                                                                    150
 p=sqrt(2.*K*TEMP);
 do{
  g=r-0.5*(1+erf(u));
  g1=0.56419*exp(-u*u);
  u+=g/g1;
  i++;
 }
 while(i<20);
 return(p*u);
}                                                                   160
#define MAXSTP 1000
#define TINY 1.0e-30
simulate(DT,fpo)
 float DT;
 char *fpo;
{
 int i,ii,in,j,k,n,nstp;
 float dumy,dxsav,h,hdid,hnext,side[3],x=0.,xsav;
```

```
float en[3], es[3],yst[2][3][NA],yn[2][3][NA];
float ystart[2][3][NA];                                          170
h=DT;


for(in=0; in<=1; in++){
 for(n=1;n<=N;n++){
  for(j=0;j<=2;j++){
   ystart[in][j][n]=y[in][j][n];
  }
 }
}
print1(fpo,x);                                                   180
for(nstp=1; nstp<=MAXSTP;nstp++){
 for(i=0;i<=2;i++){
  es[i]=en[i];
 }
 x+=h;
 turev(dy,y);
 for(n=1;n<=N;n++){
  for(in=0;in<=1;in++){
   for(j=0;j<=2;j++){
    yn[in][j][n]=y[in][j][n];                                    190
```

```
    y[in][j][n]+=1./6.*h*dy[in][j][n];
    yst[in][j][n]=yn[in][j][n]+0.5*h*dy[in][j][n];
   }
  }
 }
turev(dy,yst);
for(n=1;n<=N;n++){
 for(in=0;in<=1;in++){
  for(j=0;j<=2;j++){
    y[in][j][n]+=1./3.*h*dy[in][j][n];                                    200
    yst[in][j][n]=yn[in][j][n]+0.5*h*dy[in][j][n];
   }
  }
 }
turev(dy,yst);
for(n=1;n<=N;n++){
 for(in=0;in<=1;in++){
  for(j=0;j<=2;j++){
    y[in][j][n]+=1./3.*h*dy[in][j][n];
    yst[in][j][n]=yn[in][j][n]+h*dy[in][j][n];                            210
   }
  }
```

```
  }
  turev(dy,yst);
  for(n=1;n<=N;n++){
   for(in=0;in<=1;in++){
    for(j=0;j<=2;j++){
     y[in][j][n]+=1./6.*h*dy[in][j][n];
    }
   }                                                    220
   fprintf(fpo,"\n");
  }
  print2(fpo,x);
  for(n=1;n<=N;n++){
   for(j=0;j<=2;j++){
    fprintf(fpo,"%5.3f ",y[0][j][n]);
   }
   fprintf(fpo,"\n");
  }
  periodic();                                           230
 }
}
periodic()
{
```

```
int i,j,k,n;
for(j=0;j<=2;j++){
 for(n=1;n<=N;n++){
  if(y[0][j][n]< -0.05*side[j]){
   printf("kucuk\n");
   k=abs(y[0][j][n]/side[j]);                                    240
   y[0][j][n]+=side[j]*(k+1);
  }
  if(y[0][j][n]>1.05*side[j]){
   printf("buyuk\n");
   k=y[0][j][n]/side[j];
   y[0][j][n]-=side[j]*k;
  }
 }
}                                                                250
turev(dy,y)
 float dy[][3][NA],y[][3][NA];
{
 int n,i,j,k;
 for(n=1;n<=N;n++){
  for(j=0;j<=2;j++){
```

```
   dy[0][j][n]=y[1][j][n];
  }
 }
 forces();                                                    260
}
forces()
{
 int dc[NB],i,k,n1, n2;
 float f[3],m[3],sum=0.,f1,f2,a,Etot(),Ebs();
 float ds[NB],dumy=0, hue[4],r, x[3][27];
 matrix(ds);
 sort(dc, ds,4*N);
 for(n1=1; n1<=N; n1++){
  for(k=0; k<=2; k++){                                        270
   dy[1][k][n1]=0.;
  }
 }
 for(n1=1; n1<=N-1; n1++){
  for(n2=n1+1; n2<=N; n2++){
   hiroko(n1,n2,x);
   indivf(dc,ds,m,n1,n2,x);
   for(i=0;i<=26;i++){
```

```
 r=sqrt(x[0][i]*x[0][i]+x[1][i]*x[1][i]+x[2][i]*x[2][i]);
 if(r<3.3 && r>.1){                                                    280
  dumy=0.5*(ter[1]+2.*ter[2]*r+3.*ter[3]*r*r);
  for(k=0;k<=2;k++){
   dy[1][k][n1]+=dumy*x[k][i]/r;
   dy[1][k][n2]-=dumy*x[k][i]/r;
   }
  }
 }
}
for(k=0;k<=2;k++){
 dy[1][k][n1]-=m[k];                                                   290
 dy[1][k][n2]+=m[k];
}
if(r<3.5 && r>3.3){
  dumy=3.*1196.300446*r*r+2.*(-12194.395773)*r+41396.729024;
  for(k=0;k<=2;k++){
   dy[1][k][n1]+=dumy*x[k][i]/r;
   dy[1][k][n2]-=dumy*x[k][i]/r;
   dy[1][k][n1]-=m[k];
   dy[1][k][n2]+=m[k];
  }                                                                    300
```

```
    }
   }
  }
 }
}
indivf(dc,ds,m,n1,n2,x)
 float ds[],m[],x[][27];
 int dc[],n1,n2;
{
 int e,fp,sp,i,j,k;                                          310
 float a,dm[3][4][4],r;
 fp=4*(n1-1)+1;
 sp=4*(n2-1)+1;
 for(k=0;k<=2;k++){
  m[k]=0.;
 }
 if(r>3.5){
  for(k=0;k<=2;k++){
   for(i=0;i<=3;i++){
    for(j=0;j<=3;j++){                                       320
     dm[k][i][j]=0.;
    }
```

```
  }
 }
 return;
}
calsum(dm,m,x);
if(r>3.3 && r<3.5)calcf2(dm,m,n1,n2);
for(e=1;e<=2*N;e++){
 for(i=0;i<=3;i++){                                                    330
  for(j=0;j<=3;j++){
   for(k=0;k<=2;k++){
    m[k]+=h[i+fp][dc[e]]*h[j+sp][dc[e]]*dm[k][i][j];
   }
  }
 }
}
}
calsum(dm,m,x)
 float dm[][4][4],m[],x[][27];                                         340
{
 int i,j,k,m1,m2;
 float dmt[3][4][4],p[3],r2;
```

```
 for(i=0;i<=26;i++){
  for(k=0;k<=2;k++){
   p[k]=x[k][i];
   r2+=p[k]*p[k];
  }
  calcf(dmt,m,r2,p);                                          350
  for(m1=0;j<=3;j++){
   for(m2=0;m2<=3;m2++){
    for(k=0;k<=2;k++){
     dm[k][m1][m2]+=dmt[k][m1][m2];
    }
   }
  }
 }
}
                                                              360
calcf(dm,m,r2,x)
 float dm[][4][4],m[],r2,x[];
{
 int i,j,k,l,m;
 float r,r4,r5,r6;
```

```
r=sqrt(r2);
r4=r2*r2;
r5=r*r4;
r6=r*r5;                                                    370
if(r2<6){
 for(k=0; k<=2; k++){
  dm[k][0][0]=2.*mat[0]*x[k]/r4;
  for(l=1; l<=3; l++){
   if(k==l-1){
    dm[k][0][l]=(3.*pow(x[l-1],2.)-r2)/r5*mat[1];
   }
   if(k!=l-1){
    dm[k][0][l]=(3.*x[l-1]*x[k])/r5*mat[1];
    dm[k][l][0]=(-dm[k][0][l]);                             380
   }
   if(l==k+1){
    dm[k][l][l]=(4.*pow(x[k], 3.)-2.*x[k]*r2)/r6*(mat[2]-mat[3])+2.*x[k]/r4*mat[3];
   }
   if(l!=k+1){
    dm[k][l][l]=4.*pow(x[l-1], 2.)*x[k]/r6*(mat[2]-mat[3])+2.*x[k]/r4*mat[3];
   }
  }
```

```
  }
  dm[0][1][2]=(4.*x[0]*x[0]*x[1]-x[1]*r2)/r6*(mat[2]-mat[3]);                    390
  dm[1][1][2]=(4.*x[0]*x[1]*x[1]-x[0]*r2)/r6*(mat[2]-mat[3]);
  dm[2][1][2]=(4.*x[0]*x[1]*x[2])/r6*(mat[2]-mat[3]);
  dm[0][1][3]=(4.*x[0]*x[0]*x[2]-x[2]*r2)/r6*(mat[2]-mat[3]);
  dm[1][1][3]=(4.*x[0]*x[1]*x[2])/r6*(mat[2]-mat[3]);
  dm[2][1][3]=(4.*x[0]*x[2]*x[2]-x[0]*r2)/r6*(mat[2]-mat[3]);
  dm[0][2][3]=(4.*x[0]*x[1]*x[2])/r6*(mat[2]-mat[3]);
  dm[1][2][3]=(4.*x[1]*x[1]*x[2]-x[2]*r2)/r6*(mat[2]-mat[3]);
  dm[2][2][3]=(4.*x[1]*x[2]*x[2]-x[1]*r2)/r6*(mat[2]-mat[3]);
 }
 for(k=0; k<=2; k++){                                                           400
  dm[k][2][1]=dm[k][1][2];
  dm[k][3][2]=dm[k][2][3];
  dm[k][3][1]=dm[k][1][3];
 }
}
calcf2(dm,m,n1,n2)
 float dm[][4][4];
 int n1,n2;
{
 int i,j,k,l,m1,m2;                                                             410
```

```
 float a,b,c,elem(),hue[4],r,x[3],side[3];
 float judy1,judy2;
 r=sqrt(hiroko(n1,n2,x));
 judy1=inter5[0]*r*r*r*r*r+inter5[1]*r*r*r*r+inter5[2]*r*r*r+inter5[3]*r*r;
 judy2=5.*inter5[0]+4.*inter5[1]/r+3.*inter5[2]/(r*r)+2.*inter5[3]/(r*r*r);
 calcf(dm,m,n1,n2);
 for(m1=0;m1<=3;m1++){
  for(m2=0;m2<=3;m2++){
   for(k=0;k<=2;k++){
    dm[k][m1][m2]=judy1*dm[k][m1][m2]+elem(m1,m2,n1,n2)*x[k]/r*judy2;          420
   }
  }
 }
}
matrix(ds)
 float ds[];
{
 int i,j,k,n1,n2,m1,m2;
 float dir[3],elem(),d02=5.5815,dss[NB],r2,dumy;
 float x[3][27],telem();                                                      430
 i=0;
 for(n1=1; n1<=N; n1++){
```

```
 for(n2=1; n2<=N; n2++){
  hiroko(n1,n2,x);
  for(m1=0; m1<=3; m1++){
   for(m2=0; m2<=3; m2++){
    h[(n1-1)*4+m1+1][(n2-1)*4+m2+1]=telem(m1,m2,x);
    printf("%4.1f ",h[i][j]);
    j++;
   }                                                    440
  }
  printf("\n");
 }
}
 dia(ds);
}
float telem(m1,m2,x)
 int m1,m2;
 float x[][27];
{                                                       450
 int i,j,k;
 float dummy=0,p[3];
 for(i=0;i<=26;i++){
  for(j=0;j<=2;j++){
```

```
   p[j]=x[j][i];
  }
  dummy+=elem(m1,m2,p);
 }
 return dummy;
}                                                                    460
float elem(m1,m2,p)
 int m1,m2;
 float p[];
{
 int i,j,k,kk,l,m,n,ni=1;
 float c[3], r2, r,r2i;
 float hue[4],x[3];
 r2=p[0]*p[0]+p[1]*p[1]+p[2]*p[2];
 r=sqrt(r2);
 if(r2==0. && m1==m2 && m1==0) return Es;                            470
 if(r2==0. && m1==m2 && m1!=0) return Ep;
 if(r2==0. && m1!=m2) return 0.;


 if(r>3.5) return 0.;
 c[0]=p[0]/r;
 c[1]=p[1]/r;
```

```
 c[2]=p[2]/r;
 if(r<3.3){
  if(m1==m2 && m1==0) return mat[0]/r2;
  if(m1==0 && m2!=0) return c[m2-1]*mat[1]/r2;                    480
  if(m1!=0 && m2==0) return -c[m1-1]*mat[1]/r2;
  if(m1==m2 && m2!=0) return c[m1-1]*c[m1-1]/r2*(mat[2]-mat[3])+mat[3]/r2;
  if(m1!=m2 && m1!=0 && m2!=0) return c[m1-1]*c[m2-1]/r2*(mat[2]-mat[3]);
 }
  r2i=inter5[0]*r*r2+inter5[1]*r2+inter5[2]*r+inter5[3];
  if(m1==m2 && m1==0) return mat[0]*r2i;
  if(m1==0 && m2!=0) return c[m2-1]*mat[1]*r2i;
  if(m1!=0 && m2==0) return -c[m1-1]*mat[1]*r2i;
  if(m1==m2 && m2!=0) return c[m1-1]*c[m1-1]*r2i*(mat[2]-mat[3])+mat[3]*r2i;
  if(m1!=m2 && m1!=0 && m2!=0) return c[m1-1]*c[m2-1]*r2i*(mat[2]-mat[3]);  490
}


hiroko(n1,n2,x)
 int n1, n2;
 float x[][27];
{
 int i=0,k,l,m,n;
 float c[3],r;
```

```
 printf("n1=%d n2=%d \n",n1,n2);
 for(l=(-1);l<=1;l++){                                          500
  c[0]=side[0]*l;
  for(n=(-1);n<=1;n++){
   c[1]=side[1]*n;
   for(m=(-1);m<=1;m++){
    c[2]=side[2]*m;
    for(k=0;k<=2;k++){
     x[k][i]=y[0][k][n2]+c[k]-y[0][k][n1];
    }
    r=x[0][i]*x[0][i]+x[1][i]*x[1][i]+x[2][i]*x[2][i];
    if(r<6 && r>5)printf("%d %d %6.3f\n",n1,n2,i,r);            510
    i++;
   }
  }
 }
}


dia(ds)
 float ds[];
{
 int i, j, n=4*N,np=50;                                         520
```

```
 float d[NB], e[NB],ve,sum=0.;
 tred2(h,n,d,e);
 tqli(d,e,n,h);
 for(i=1; i<=n; i++){
  ds[i]=d[i];
 }
}
tred2(a,n,d,e)
float a[][NB] ,d[],e[];
int n;                                                          530
{
 int l,k,j,i;
 float scale,hh,h,g,f;
 for (i=n;i>=2;i--) {
 l=i-1;
 h=scale=0.0;
 if (l > 1) {
  for (k=1;k<=l;k++)
   scale += fabs(a[i][k]);
   if (scale == 0.0)                                            540
    e[i]=a[i][l];
   else {
```

```
for (k=1;k<=l;k++) {
 a[i][k] /= scale;
 h += a[i][k]*a[i][k];
}
f=a[i][l];
g = f>0 ? -sqrt(h) : sqrt(h);
e[i]=scale*g;
h -= f*g;
a[i][l]=f-g;
f=0.0;
for (j=1;j<=l;j++) {
 a[j][i]=a[i][j]/h;
 g=0.0;
 for (k=1;k<=j;k++)
  g += a[j][k]*a[i][k];
  for (k=j+1;k<=l;k++)
   g += a[k][j]*a[i][k];
   e[j]=g/h;
   f += e[j]*a[i][j];
  }
  hh=f/(h+h);
  for (j=1;j<=l;j++) {
```

```
    f=a[i][j];
    e[j]=g=e[j]-hh*f;
    for (k=1;k<=j;k++)
     a[j][k] -= (f*e[k]+g*a[i][k]);
    }
   }                                                    570
  } else
  e[i]=a[i][l];
  d[i]=h;
 }
 d[1]=0.0;
 e[1]=0.0;
 for (i=1;i<=n;i++) {
  l=i-1;
  if (d[i]) {
   for (j=1;j<=l;j++) {                                 580
    g=0.0;
    for (k=1;k<=l;k++)
     g += a[i][k]*a[k][j];
        for (k=1;k<=l;k++)
         a[k][j] -= g*a[k][i];
        }
```

```
        }
        d[i]=a[i][i];
        a[i][i]=1.0;
        for (j=1;j<=l;j++) a[j][i]=a[i][j]=0.0;                    590
            }
}

#define SIGN(a,b) ((b)<0 ? -fabs(a) : fabs(a))

tqli(d,e,n,z)
float d[],e[],z[][NB];
int n;
{
 int m,l,iter,i,k;                                                 600
 float s,r,p,g,f,dd,c,b;

 for (i=2;i<=n;i++) e[i-1]=e[i];
  e[n]=0.0;
  for (l=1;l<=n;l++) {
   iter=0;
   do {
   for (m=l;m<=n-1;m++) {
```

```
 dd=fabs(d[m])+fabs(d[m+1]);
 if ((float)(fabs(e[m])+dd) == dd) break;                    610
}
if (m != l) {
 if (iter++ == 30);
 g=(d[l+1]-d[l])/(2.0*e[l]);
 r=sqrt((g*g)+1.0);
 g=d[m]-d[l]+e[l]/(g+SIGN(r,g));
 s=c=1.0;
 p=0.0;
 for (i=m-1;i>=l;i--) {
  f=s*e[i];                                                  620
  b=c*e[i];
  if (fabs(f) >= fabs(g)) {
   c=g/f;
   r=sqrt((c*c)+1.0);
   e[i+1]=f*r;
   c *= (s=1.0/r);
  } else {
  s=f/g;
  r=sqrt((s*s)+1.0);
  e[i+1]=g*r;                                                630
```

```
      s *= (c=1.0/r);
    }
    g=d[i+1]-p;
    r=(d[i]-g)*s+2.0*c*b;
    p=s*r;
    d[i+1]=g+p;
    g=c*p;
    for (k=1;k<=n;k++) {
     f=z[k][i+1];
     z[k][i+1]=s*z[k][i]+c*f;                                    640
     z[k][i]=c*z[k][i]-s*f;
    }
   }
   d[l]=d[l]-p;
   e[l]=g;
   e[m]=0.0;
  }
 } while (m != l);
}
}                                                                650
sort(dc,ds,n)
 int dc[], n;
```

```
 float ds[];
{
 int i, inter2, j;
 float inter;
 for(i=1; i<=n; i++){
  dc[i]=i;
 }
 for(i=2; i<=n; i++){                                    660
  for(j=1; j<i; j++){
   if(ds[i]<ds[j]){
    inter=ds[i];
    ds[i]=ds[j];
    ds[j]=inter;
    inter2=dc[i];
    dc[i]=dc[j];
    dc[j]=inter2;
   }
  }                                                      670
 }
}
energy(e)
 float e[];
```

```
{
int dc[NB],i,j,k,m,am=1,n1,n2,l;
float BS,ds[NB],f1,f2,h[NB][NB],hue[4],r=0., r1, r2, sum=0.,c[3], x[3][27];
float Etot(),Ebs(),de=0.;
e[0]=e[1]=e[2]=0.;
matrix(ds);                                                    680
sort(dc,ds,4*N);
for(i=1; i<=2.*N;i++){
 sum+=ds[i];
}
BS=2.*sum/N;
e[0]+=BS;
for(i=1;i<=N-1;i++){
 for(j=i+1;j<=N;j++){
  hiroko(i,j,x);
  for(k=0;k<=26;k++){                                          690
   r=sqrt(x[0][k]*x[0][k]+x[1][k]*x[1][k]+x[2][k]*x[2][k]);
   if(r<3.5){
    if(r<3.3){
     de+=0.5*(ter[0]+ter[1]*r+ter[2]*r*r+ter[3]*r*r*r);
    }
    if(r<3.5 && r>3.3){
```

```
    /* de+=1196.300446*r*r*r+(-12194.395773)*r*r+41396.729024*r-46798.584980;*/
     }
    }
   }                                                                    700
  }
 }
 e[0]+=de/N;
 for(i=1; i<=N; i++){
  e[1]+=0.5*(y[1][1][i]*y[1][1][i]+y[1][2][i]*y[1][2][i]+y[1][0][i]*y[1][0][i]); }
 e[1]/=N;
 e[2]=e[0]+e[1];
}
float es[3];
print1(fpo,t)                                                           710
  float t;
  char *fpo;
{
  int i,in,j,k,n;
  float en[3];
  energy(en);
  for(i=0;i<=2;i++){
   es[i]=en[i];
```

```
    }
    printf("   TIME        P.E         K.E         T.E         K.D         P.D      T.D \n");
    printf("\n %8.5f    %8.5f    %8.5f    %8.5f  %8.5f  %8.5f  %8.5f\n",
     t,en[0],en[1],en[2],(en[1]-en[1]),(en[0]-en[0]),
     (en[2]-en[2])/fabs(en[2])*100);
    /*fprintf(fpo," TIME      P.E       K.E       T.E       K.D       P.D       T.D \n");
    fprintf(fpo,"\n %8.5f   %8.5f   %8.5f   %8.5f  %8.5f  %8.5f  %8.5f\n",
     t,en[0],en[1],en[2],(en[1]-en[1]),(en[0]-en[0]),
     (en[2]-en[2])/fabs(en[2])*100);*/
}
print2(fpo,t)
   float t;
   char *fpo;
{
   int i,in,j,k,n;
   float en[3];
   energy(en);
   printf("\n%8.5f    %8.5f    %8.5f    %8.5f  %8.5f  %8.5f  %8.5f\n",t,en[0],en[1],en[2],
            (en[1]-es[1]),(en[0]-es[0]),(en[2]-es[2])/fabs(es[2])*100);
   /*fprintf(fpo,"\n%8.5f   %8.5f   %8.5f   %8.5f  %8.5f  %8.5f  %8.5f\n",t,en[0],en[1],en[2],
             (en[1]-es[1]),(en[0]-es[0]),(en[2]-es[2])/fabs(es[2])*100);*/
}
```

```
float ext(f1,f2,hue,r)
float f1,f2,hue[],r;
{
 float ex[4],k=3.3,l=3.5;
 exco(ex,f1,f2,hue,k,l);
 return ex[0]*r*r*r+ex[1]*r*r+ex[2]*r+ex[3];
}
exco(ex,f1,f2,hue,k,l)
float ex[],f1,f2,hue[],k,l;
{                                                          750
 float p;

 p=k*k*k-l*l*l+3.*k*l*(l-k);
 hue[0]=ex[0]=(-2.*f1+f2*(k-l))/p;
 hue[1]=ex[1]=(-1)*(f1*(-3.*k-3.*l)+f2*(k*k+k*l-2*l*l))/p;
 hue[2]=ex[2]=l*(-f1*6.*k+f2*(2.*k*k-k*l-l*l))/p;
 ex[3]=(-1)*l*l/p*(f1*(l-3.*k)+f2*(-k*l+k*k));
}

float Etot(r)                                              760
 float r;
{
```

```
 return E0*(1+(r-r0)/AA)*exp(-(r-r0)/AA);
}
float Ebs(r)
 float r;
{
 return A0+A1*(r-rb)+A2*(r-rb)*(r-rb)+A3*pow((r-rb),3.);
}
```

770

780